10/519085
DT01 Rec'd PCT/PTO 23 DEC 2004
710.1011

IN THE UNITED STATES PATENT & TRADEMARK OFFICE

Re:	Application of:	Johann Ulrich ZIMMERMANN
	Serial No.:	To Be Assigned
	Filed:	Herewith as national phase of International Patent Application PCT/EP2003/006270, filed June 13, 2003
	For:	INFORMATION GENERATION SYSTEM FOR PRODUCT FORMATION

Mail Stop: PCT
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

December 23, 2004

LETTER RE: PRIORITY

Sir:

Applicant hereby claims priority of German Applications Serial No. DE 102 28 906.9, filed June 27, 2002 and DE 102 36 384.6, filed August 8, 2002 through International Patent Application Serial No. PCT/EP2003/006270, filed June 13, 2003.

Respectfully submitted,

DAVIDSON, DAVIDSON & KAPPEL, LLC

By [illegible]
Robert J. Paradiso, Reg. No. 41,240
(signing for Thomas P. Canty, Reg. No. 44,586)

Davidson, Davidson & Kappel, LLC
485 Seventh Avenue, 14th Floor
New York, New York 10018
(212) 736-1940

PCT/EP 03 / 06270

BUNDESREPUBLIK DEUTSCHLAND

19. 07. 2003

Prioritätsbescheinigung über die Einreichung einer Patentanmeldung

Aktenzeichen:	102 28 906.9
Anmeldetag:	27. Juni 2002
Anmelder/Inhaber:	DaimlerChrysler AG, Stuttgart/DE
Bezeichnung:	ULEO – Universal linking of engineering objects
IPC:	G 05 B 17/00

Die angehefteten Stücke sind eine richtige und genaue Wiedergabe der ursprünglichen Unterlagen dieser Patentanmeldung.

München, den 30. Juni 2003
Deutsches Patent- und Markenamt
Der Präsident
Im Auftrag

[illegible]

Jerofsky

A 9161
06/00
EDV-L

ULEO – Universal Linking of Engineering Objects

The resulting ULEO approach aims at enabling a high-quality flow of information between applications and at universal automation of product model generation. Amongst other system characteristics, this is achieved by modeling all classes of relevant objects within a Unified Model of Engineering Objects (UMEO). All types of relationships are modeled inside a dedicated meta taxonomy of relation types and materialize inside the UMEO. Informational relations represent ontological knowledge about dependencies between object classes and are suited for cross-linking of product models. Generative relations describe knowledge on how to create instances automatically.

Each application along the product development process chain (**process step application - ProSAp**) has to fulfill specific tasks using specialized technical knowledge and modi operandi for problem solving. As a result, each ProSAp has its own view of the product. IT solutions must adapt to these different characteristics in order to be able to represent effective and intuitive tools and aids for the users. Feature types should correspond to the concepts the users have in mind while performing their tasks. Features assume the role of building blocks for creating view-specific product models. While domain-specific feature types have a variety of benefits, there is also a down-side: the use of specialized feature types may lead to a logical separation of the feature-based models produced. Known approaches covering **feature mapping** functionality – also called **feature conversion** or **feature transformation** systems – try to tackle this problem in various manners, all based on the same underlying principle: a transformation between two feature sets A and B is performed by generating the new set of feature instances B from the given one A. In the general case there may be n-to-m relations between the features of A and B.

Pure feature mapping doesn't really close the gap between feature-based models, since it targets the generation aspect, neglecting what happens afterwards. It "jumps" the gap instead of closing it. To cover these new aspects, the concept of **feature linking** will be introduced. Feature linking is informally defined here as feature mapping together with the generation and maintenance of persistent links between the mapped feature instances. This approaches aims at finding a solution for feature linking, thereby enabling a high-quality flow of information between applications and a universal automation of product model generation.

STATE OF THE ART

Systems are known which convert feature-based models indirectly via an intermediate model that contains geometry-oriented, application-neutral intermediate feature classes and instances. The term *geometry-oriented* as used here means that every intermediate feature focuses on representing product geometry, although it may also contain other information. Deviating from such neutral-model approaches, other approaches use the design feature model as the central intermediate model. Instead of performing real feature mapping, they use a two-step feature recognition approach to proceed from the design feature model to other, application-specific models. Yet, also this method allows changes inside the individual views and propagates them back to the intermediate model via constraints that relate geometric entities. The key advantage of intermediate-model approaches is that they reduce the number of necessary mappings between the set of feature-based applications. However, intermediate-systems also have approach-inherent drawbacks: they suffer from a certain lack of flexibility in that the set of expressible information is restricted to the expressiveness of the intermediate model. This model has to be able to represent all kinds of information which is relevant along the process chain. As a consequence, intermediate features may be considered all-purpose feature classes, which is cumbersome and leads to well-known effects of non-normalized data storage, such as redundancy and partly inappropriate attributes. Another principle drawback of such approaches is that they inhibit expressing direct relations between features (or feature classes) of different applications.

Characteristic: How Features Are Modeled

One key set of characteristics of feature-based systems describes the way in which the features are modeled. In this respect, most publications are rather vague about this, thus forcing the reader to make assumptions about most of the details of the models used inside the investigated approaches. It seems to have become standard in more recent approaches to classify features and to differentiate between **feature classes and instances** accordingly as well as to use **taxonomies** to arrange the feature classes. A taxonomy is a hierarchy of feature classes based on inheritance relations. It is known to use separate taxonomies of feature classes for the individual feature-based applications.

Characteristic: How Mapping Knowledge Is Modeled

Current approaches also differ in the way they model knowledge about mapping. Looking at **where** this information is stored, the literature yields several alternatives: (a) inside or (b) outside the feature definitions (classes). If outside, mapping knowledge could be represented (b1) in hard code inside the system's algorithm or (b2) stored inside a separate knowledge base. Storing mapping knowledge inside a feature class, alternative (a), means to have it partly where it belongs, and partly not, as mapping always occurs between at least two applications. Another shortcoming is the need to change the feature classes themselves every time a destination ProSAp is added or removed from the process chain. Although these characteristics may appear to be not quite intuitive, solution (a) leads to a focused and clear representation, which enhances readability and maintenance. A representation language called Part Design Graph Language (PDGL) is known to express the knowledge inside the feature classes. The most straightforward solution, method (b1), is used, for example, by mapping user-defined design features on generic machining features via a hard-coded Constructive Feature Tree (CFT) reconstruction algorithm. It is obvious that hard-coded solutions lack flexibility and are more difficult to understand for the programmers and users of the system. A system according to the approach (b2) interprets production rules to map them onto machining process operations by choosing alternatives from a pre-defined machining process graph. Sometimes rules are used to weight mapping hints inside design features. One approach is to convert neutral features into application-specific features using production rules. Generally, rule-based approaches, just like any other knowledge-based approach, are very flexible in the sense that the mapping knowledge can be adjusted without changing the program, which can also be done by non-programmers. This is essential for practical use of a system since knowledge may change over time, and information about the handling of new feature types may have to be added. Additionally, because all available knowledge is stored within a single location, its consistency can be established quite easily. On the other hand, this property leads to comprehensibility problems, which make isolated knowledge bases – particularly large ones – hard to maintain. Hybrid approaches are, for example, the hint-based systems, which store some of the information relevant for feature mapping directly inside the feature instances (a) and another part elsewhere, for instance hard-coded inside the program code (b1). For example, "codes" inside a design-feature-based model are used to derive downstream models.

Characteristic: Generation and Maintenance of Links Between Feature Sets

Further known examples of intermediate-model approaches claim to allow designers and application experts to communicate via their intermediate models. Yet because direct links between ProSAp models are missing which could bypass the intermediate model, this kind communication appears to be restricted to features representing some kind of geometry (geometry-centered communication). It is not possible to directly link a feature to any other one or to any non-feature object at all. One approach uses notions like *feature linking* and *inter feature links*, but their meaning also seems to be geometry-focused. It is not mentioned how the links are established.

Other Characteristics

To conclude, existing commercial as well as scientific solutions focus on automation aspects, seeing ProSAp integration more or less solely on the periphery – both in terms of the set of object types they are able to interconnect, as well as in terms of the kind of information the interconnections can convey. In terms of automation capabilities offered, known approaches tend to address specialized areas. There is no system that is generally applicable for the whole range of objects relevant in engineering. In this sense, state-of-the-art systems do not yet close the informational gap between process steps to the degree desired by engineers.

2[illegible] ULEO APPROACH

The next paragraphs will introduce the approach called Universal Linking of Engineering Objects (ULEO). **Engineering objects** (EOs) are introduced here as any objects relevant in engineering, such as assemblies, features, parts, surfaces, tolerances, materials, etc.

Characteristic: Direct or indirect Mapping

The ability to model relations between relevant objects and/or classes directly seems to be a prerequisite for sophisticated process chain integration. Hence, a solution **which does not use intermediate models** is proposed.

Characteristic: How Features Are Modeled

The capability to model all classes of features and other relevant objects inside a single global model is a characteristic that enables to relate these entities to each other in a clear and straightforward fashion and to reason about them within the same context. A bit less formally put one could say that these entities know about each other.

None of the known systems provides this characteristic, they get by without global taxonomies. Also, none of the known approaches models feature classes and classes of other objects inside the same taxonomy.

Characteristic: How Mapping Knowledge Is Modeled

The combination of methods for storage of knowledge about mapping within special relations (which inter-connect the feature classes inside the class taxonomy (b3)), with the use of conceptual graphs seems to avoid the aforementioned shortcomings of the systems set out above. The knowledge is preferably stored in a highly targeted way, directly related to the objects it refers to. This makes it universally usable, flexible, intuitive, and easy to read and to maintain (edit, add, delete elements).

A preferred characteristic of ULEO is to store this information within special relations which link the feature classes inside the class taxonomy (b3). This solution can be regarded as a special kind of knowledge base formed by a structured network tied to the feature classes as a second information layer. Any combination of the mentioned principle alternatives is also conceivable.

Conceptual graphs are an option for handling information about objects and their inter-relations in a clear fashion.

Characteristic: Generation and Maintenance of Links between Feature Sets

A further characteristic of feature mapping systems is the way of generation and maintenance of links between the source and destination feature sets. In other words, does a system record its mappings and does it create dedicated and persistent links of some kind between the respective objects? This aspect is not solved by the known approaches.

Links between feature sets are the carriers of information for inter-ProSAp communication. In this respect, existing feature-based systems are also not able to offer a solution.

Other Characteristics

Although there are other characteristics which are fundamental for the motivation of the ULEO approach, they will just be touched on here. For example, the following aspects are tackled by the ULEO approach: **knowledge processing**, modeling of **feature constellations**, **complexity** and **cardinality** of possible **relations** between the mapped feature sets, **classified relations**, support of **user-defined features**, aspects of **concurrent engineering**, **mapping on demand vs. mapping on the fly**.

2.2 Main Thesis

The goals specified above can be reached by modeling engineering objects and their corresponding interrelations as follows (see Figure 1):

(1) All EO classes are modeled within a central taxonomy called **Unified Model of Engineering Objects (UMEO)**. "Central" means that there is only one taxonomy for all ProSAps, and the taxonomy is stored in only one globally accessible location.

(2) All **types** of relationships between EO classes and/or instances are modeled inside a dedicated **meta taxonomy of relation types (MTRT)**, which is situated logically one abstraction layer above UMEO. Relation types can be inheritance, aggregation or **engineering object relations (EORs)**, which are specialized directed associations. MTRT covers **informational** EORs (**IEORs**) as well as **generative** EORs (**GEORs**).

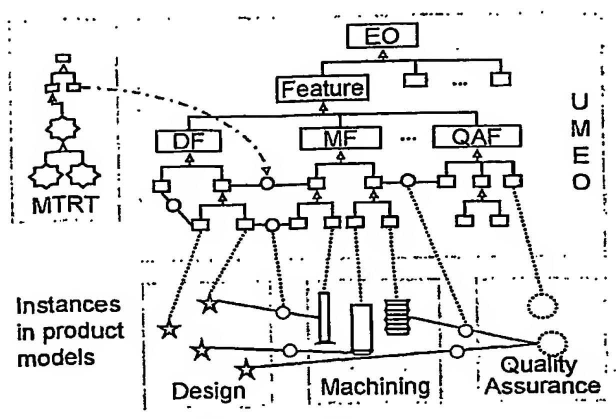


Figure 1: UMEO classes and instances.

2.3 Unified Model of Engineering Objects and Engineering Object Relations

UMEO has the ability to allow all kinds of application-specific, standardized, and non-standardized EOs to be arranged in a unique and globally accessible taxonomy. This ensures that all the applications accessing the model get the same information and use the same concepts. Through the common model they all use the same notions. In addition, the set of expressible relations is no longer restricted to inter-feature relations. Instead, relations between feature components (intra-feature links) are also representable as well as relations between features and other entities like parts, assemblies, and resources. This increases the value of the model significantly as these relations facilitate assembly modeling (e.g. assembly features) and assignment of resources and processes, which are key milestones in building a bridge to the logistic process chain. Feature linking is generalized into a universal linking of engineering objects. Information on relations between objects or object classes is modeled in terms of dedicated relation objects, denoted as round black or white symbols in Figure 1. This permits relation-centered information to be kept outside the related objects or object classes. Thus, relations between the latter may change, while the related EOs and EO classes remain unchanged. Also, each EOR is of a specific type. In order to structure the possibly large quantity of EOR types, a special MTRT is introduced in a preferred embodiment: its elements materialize inside UMEO as relations on the EO class level and instantiate inside the individual product models as relations on the EO level. As a result, cross-linked product models arise (see lower third of Figure 1). As EOR materializations are preferably an integral part of UMEO, they are also inherited from parent to child EO classes, so that models remain as clear as possible. MTRT allows new relation types to be easily derived from existing ones using inheritance; and existing service applications such as cost estimation can roughly estimate the semantics of derived relation types by knowing their parent classes. The two main types of EORs cited above, *informational EORs* and *generative EORs* will be introduced. Both enable structured and targeted modeling of knowledge, so that it is directly related to the respective classes and objects inside the model. This matches the feature principle of attaching information exactly to what it refers to, thereby avoiding unintelligible all-in-one and hard-to-maintain knowledge bases. Moreover, using inheritance together with EO classes makes it possible to apply EORs which reference some parent EO class also to its child classes. This is an elegant way of deducing new, explicit knowledge from the implicit. General knowledge is attached to parent classes and exceptions to their children.

Informational EORs (IEORs) describe logical relationships between EO classes, e.g. a relation between *n* design features and *m* machining features, which might be called "is_machined_as" and could describe which machining features are necessary to machine the related design features. Moreover, with IEORs, UMEO may be augmented gradually by ontological knowledge, thus adding sophisticated information about the EOs' semantics, usage, and embedding within extended contexts of other applications' knowledge, for example. EO classes and IEOR classes can be instantiated into application-specific product models. IEOR instances interrelate individual product models by linking the logically corresponding EO instances inside of them, permitting information to be interchanged between ProSAps on the feature level. Hence, IEORs have the potential to facilitate integration of the applications along the process chain on the information level. They may be used by application programs for a wide variety of purposes, such as design-to-X (for example cost estimation, manufacturability checks), feedback from downstream to upstream applications, etc.

Generative EORs (*GEORs*) describe knowledge on how to instantiate EOs and EORs automatically, for example triggered by and taking into account conditions inside product models. Examples of the latter are prior instantiations of features and their parameter values. Interpretation of these links results in automation of engineering tasks: thus, GEORs represent generative engineering knowledge. They provide a functionality subsuming feature mapping but are, in contrast, not limited to feature classes – in fact they can consider any engineering object. After each modification of a specific product model, the linking algorithm can be triggered to locate the class of the manipulated object inside UMEO and to search for GEORs related to it. The GEORs' contents are interpreted by the system. These could be, for example, values of feature parameters or the number of new instances to be created. Since UMEO's contents are meant for many different applications, they only consist of information about what is to be done, not how this is to be done. So, linking (mapping) knowledge may be changed without having to change ProSAp-specific elements.

ULEO generalizes the concept of features to that of engineering objects and focuses on the modeling and usage of typed relations between them. Thus, problems in the fields of not only automation but also process integration can be solved by applying ULEO. Information is stored locally and directly related to objects or object classes. The result is a clearly structured and maintainable knowledge base. ULEO is an open approach and sophistication of knowledge representation inside its models is scalable, which permits both backward-compatibility to older feature-based systems and also integration of and cooperation with company-wide information systems. The approach provides the base technology for further service applications. Process step applications are enabled to broaden their informational context, providing engineers with the information necessary for well-founded decision-making. Thus, ULEO supports cooperation implicitly and explicitly, offering a rich variety of information stemming from various applications along the process chain.

For example, the following application is solved by the ULEO approach: detail design of cylinder heads and mold tools is targeted together with the corresponding machining planning. Within this context, a special embodiment of ULEO has been developed: EO constellations are sets of EO classes, related to each other through EOR classes. All members of an EO constellation can be instantiated within a single action – this also includes instantiations into several product models such as finish-part and machining model. The EO instances' parameter values are synchronized. Another application is inspection planning for the automotive body in white.

APPLYING UNIVERSAL LINKING OF ENGINEERING OBJECTS IN THE AUTOMOTIVE INDUSTRY – PRACTICAL ASPECTS, BENEFITS, AND IMPLEMENTATIONS

The bi-directional communication of CAD programs with ;ubsequent applications such as process planning remains a key ;hallenge in design-for-the-lifecycle. While it seems sensible :hat individual applications use their own collection of feature ;ypes and thereby allow users to have their specific perspective)f the product, it is still difficult to automatically close the gap)etween the variety of applications. Universal Linking of Engineering Objects (ULEO) targets this concern. It is general ;nough to facilitate informational integration, i.e. integration on the information level, of the applications along the process :hain.

This paper examines a number of applications for ex[illegible] ULEO's benefits in the field of automotive d[illegible]ent and reports on the associated software im[illegible]tations.

This contribution addresses aspects of an invention in the field of process chain integration in the automotive industry.

How features are modeled strongly influences the resulting benefits of feature technology. It has a direct impact on the quality of the information representable inside product models and materializes, in terms of the collection of feature properties, relations between features, and the meta-information on both. Other quality criteria are extensibility of models and the option to integrate them into other models, absence of redundancy, availability of building-blocks for new features, user-definability of features and relations, etc.

Modeling of features in accordance with the object-oriented paradigm seems to have become the standard method. This means that features are differentiated and arranged into feature types or classes. After that, feature classes can be instantiated into product models (→ feature instances). A feature class typically comprehends attributes (parameters), which describe the [illegible]ic properties, and sometimes also methods, which d[illegible] the dynamic behavior of the respective feature in[illegible] To reduce redundancy among feature classes, one or more steps of abstraction are typically performed where common elements of some feature classes are extracted and conglomerated to a new feature class, called the parent class, which is related to the child feature classes through a kind of relation called 'is_a' or specialization relation. Such relations are hierarchical in the sense that a parent class bequests all its properties to its child classes. The result of finding feature classes, performing abstractions, and arranging the classes by linking them through specialization relations is a taxonomy of feature classes. Feature classes on the bottom level of the taxonomy tree can be instantiated whereas classes located higher in the hierarchy cannot. The latter are also called abstract classes.

Modern feature-based applications provide engineers (hereinafter also called users) with a multitude of benefits for their work. Usually, they offer pre-defined and parameterized feature classes which can be used directly to form application-specific product models. This technique saves time, avoids errors, and reduces the number of different solutions, which in turn saves costs in manufacturing, quality assurance, and other processes. Hence, feature technology is valuable for most or, perhaps even all, of the applications comprising the development process chain: concept design, detail design, finite element calculation, tool design, manufacturing planning, assembly planning, factory layout planning, quality assurance, and disassembly planning are only some of the better known applications. Each *process step application* (*ProSAp*) has its own view of the product due to the fact that it has to fulfill special tasks using specialized technical knowledge and modi operandi for problem solving. IT solutions should adapt to this multitude of characteristics in order to be able to represent effective and intuitive tools and aids for the users. The need for an appropriate design of features and for feature-based systems subsequently arises. Features should resemble the particular concepts relevant within the respective process steps. In other words, these feature types should correspond to the concepts the users have in mind while performing their tasks, thus assigning features the role of building blocks for creating view-specific product models within the perspective of the special development task. This results in specialized sets of feature types for every ProSAp. In the following they will be called domain-specific feature types. The users of such feature-based systems take fewer mental detours when doing their job than their colleagues, who are forced to build up their task-specific concepts from lower-level elements or have to use multi-purpose concepts which match their needs only partially.

Without any further provisions, the informational contents of application-specific feature-based product models cannot be exchanged between and shared amongst applications. In order to achieve a sophisticated informational integration of applications, which includes runtime information flow, a two-level approach has to be taken.

First, the respective application-specific types, i.e. classes, of features are related to each other semantically so that the applications automatically know the meaning of the others' data and are able to interpret them accordingly. Ideal from a modeling point of view is to unify and integrate all feature classes within a universal model. This type of common vocabulary insures redundancy-free and globally reconciled usage of the classes. Thus, the informational horizons of the individual applications are widened and unified into a single, global one. The benefits inherent to object-oriented modeling, such as inheritance and overloading of parameters and methods, can also be used. An alternative is to declare the relations between separate feature models. While this supersedes the synchronization between the designers of feature-based applications regarding the protocol for accessing the globally-stored feature class information, it lacks the benefits of the unified-model approach.

The second level involves interconnecting feature-based roduct models stemming from different stages (applications) f product development. For this purpose, feature instances tored within the application-specific feature-based models are inked in such a way that those instances which describe the ame physical part/aspect of the finish product are inter-related, .g. a set of finish-part features and an according set of nachining features. In this way, information evolving as to a pecific feature instance may be passed to other process-step pplications along these inter-feature links. Once this is chieved, it becomes simpler to find out which feature nstances inside a certain product model are related to which ther feature instances inside other product models and in which way. For instance, how a design feature instance 'hole' s machined and inspected for quality will then be known. In he general case, n feature instances of ProSAp A may be related to m feature instances of ProSAp B.

In order to derive a new set of feature instances from a gi[illegible]e, *feature mapping* approaches, also called 'feature c[illegible]h', approaches perform a step-by-step processing of the [illegible]re instances in the source set, trying to apply pre-defined mapping rules, not necessarily represented as rules. Mapping rules describe which new feature instances are to be generated inside the destination set. This method, also called *feature conversion* or *feature transformation*, does not integrate feature-based applications in the sense stated above because no links between the mapped elements of the feature sets are generated and managed. This advanced approach is called *feature linking* in the following.

This section gives a brief introduction to ULEO's main concepts and to a concept called AutoCreate classes.

ULEO facilitates feature and engineering object (EO) linking. In this context, it is important to state that the above statements about features also hold true for EOs in general. Features were taken as examples because of their greater notoriety.

All kinds of application-specific, standardized, and non-standardized EOs are arranged in a unique and globally ac[illegible]e taxonomy, called the *universal model of engineering o[illegible]MEO)*. Informational and generative EO relations (*I[illegible]GEORs*) representing relations between engineering objects are used to represent ontological and generative knowledge as an integral part of UMEO. Both types of relations promote structured and focused knowledge representation, matching the feature philosophy of attaching information to exactly where it refers, thereby avoiding unintelligible all-in-one and hard-to-maintain knowledge bases. Generative EORs describe knowledge on how to create instances automatically, providing EO mapping functionality. Informational EORs describe logical relationships between EO classes, e.g. 'is_inspected_by'.

A special type of GEORs are *AutoCreate* (*AC*) classes. They are stored in UMEO with references to all classes that trigger the inherent actions (input classes) as well as all classes that are instantiated as a result of the ACs action (output classes). ACs contain the necessary information to instantiate one or more EO(R) classes into one or more product models. EO(R) means: EO or EOR. As all other UMEO contents, this information is also system independent and has to be interpreted by the application(s) performing the EO linking. Therefore, it is preferably stored as ASCII text in XML format (see below). The exact content inside the XML frame depend on the application domain, so the innermost representation language may vary.

All types of EORs are modeled in a *meta taxonomy of relation types* (*MTRT*), which structures the (possibly) large number of entries, allowing the benefits of inheritance to be reaped. MTRT's elements materialize into UMEO cross-linking EO classes. From UMEO they can be instantiated into product models linking EO instances.

Engineering objects (*EOs*) are objects relevant for product engineering, e.g. features, parts, assemblies, surfaces, and tolerances or machining steps. If EOs are modeled following the object-oriented paradigm, classes and instances can be differentiated. *Instances* are used to construct product models from the view of a special application – they represent exactly one object in the real world, the so-called domain. *Classes* are a means of grouping objects of the domain to simplify their handling. They are therefore one step higher up in abstraction than instances. Classes describe which properties are relevant and identifying for all their instances and, optionally, which values these properties may assume and which relationships correlate them with instances of other classes.

Features are considered to be objects used to model a product from the viewpoint of a certain step during product development. Products may consist of one or more assemblies which, in turn, may consist of one or more parts. Thus, features usually represent components of parts and contain some kind of geometry description (faces, edges, solids, etc.). Therefore, features are "smaller" than parts but "bigger" than solids and faces. However, depending on the application they are designed for, features may carry geometric and/or non-geometric information.

Feature constellations (*FCs*) or *combined features* (*CFs*) are groups of EOs meant to be instantiated in a single operation. They are inter-related by special EO relations (*EORs*).

Feature linking is informally defined here as feature mapping together with the generation and maintenance of persistent links between the mapped feature instances. The term is also used for EO linking.

A *user* is someone who utilizes a software application in order to fulfill a task as part of the product development process, e.g. concept design, detail design, manufacturing planning, assembly planning, etc.

User-defined features (*UDFs*) are those defined and implemented by company-specific end-users or standardization committees, without any means of programming and rebuilding the CAx system. They usually represent intellectual property of a company.

It is important to point out, that every ULEO characteristic being valid for 'features' is also valid for EOs.

ULEO may be applied in several variants. Some of them are described below.

Manual feature constellations (MFCs) are a comparatively simple approach as it is implementable in the ntext of a state-of-the-art CAD system. MFCs are geared for partial automation of EO instantiations accelerating the gineers' work and promoting avoidance of errors. The main tion is to predefine multiple EO classes and their parametric ependencies as kinds of patterns stored within dedicated braries. Such patterns can be instantiated into a product model alled the base model, which is dedicated to this task. From ere, the single elements such as features or parts can be copied with link' to their final destination models, i. e. by eans of uni-directional associations offered by the CAD ystem. Changes to single components have to be done within e base model.

Feature constellation linking (FCL) is not implementable sing current CAx systems without adding own code. No basic odel is needed and the system automatically creates copies of nstances into their final destination models. Instantiation of ultiple EO classes may occur into one or more product odels. FCL reads EOR-based information from UMEO to i[illegible] corresponding components of feature constellations, [illegible]lly perform the instantiation into the destination n[illegible]nd generate and save also those EOR instances which epresent information on their affiliation to feature onstellations. So FCL enhances capabilities for the pplications' informational integration and further automation.

General feature linking (GFL) implements all the apabilities of ULEO, adding even more functionality to FCLs. This approach is generally applicable to any single EO class or et of EO classes related on the class level to other single EO lass or sets of EO classes. *Linking on the fly* facilitates automated generation of EO instances every time specified EO classes have been instantiated. *Linking on demand* lets the system process all EO instances within a product model and perform all the pertinent actions defined in UMEO.

Cases of (a) difficult modeling due to highly complex inter-relationships between EOs, (b) mapping knowledge which is hard to retrieve from the experts, (c) the existence of a very large variety of possible solutions, or (d) a lack of desire to standardize the mapping process caused by a fear of being res[illegible] too much, e. g., might be covered by adding *feature re[illegible]n* functionality to the system. For the as yet untreated fr[illegible]f the source model, some kind of interactive *feature iden[illegible]cation* is preferably being performed: predefined feature classes are associated to geometric entities in the product model. Although associating very abstract feature classes, i. e. classes on a high taxonomic level, yields less benefit than specific ones do, it allows any geometric entity to be treated as a feature (EO). This is useful for the subsequent step: for EOs generated through feature recognition or manual feature identification, *feature linking* is performed in order to create the EOR instances necessary for a subsequent flow of information between the applications, because they inter-relate the EO instances created manually to those generated by the system. This hierarchical approach leads to maximum integration of EO-based models along the process chain combined with a maximum degree of automation performed by the software system.

In large organizations, the introduction of new information technologies in industry has a direct impact on the product creation processes as it may interrupt or interfere with the course of running projects. Hardware and software have to be installed, users have to be qualified through intensive seminars and the accompanying training. The aspect of compatibility between new and old product models also plays a key role. The more a new system differs from its predecessors, the more effort is required for training and product model conversion. For this reason, the right answer to the question of whether a new IT solution can be used in combination with the existing ones can result in large savings or additional costs.

As ULEO's modeling approach can utilize but does not presuppose a full knowledge base before being able to work sufficiently, it is universally applicable within feature-based applications and widely scalable in extension. This ranges from a featureless, isolated way of working to a cooperation of engineers provided with a broad context of information. This distinctive scalability enables evolutionary and very natural industrialization.

Application examples showed a manageable number of relevant relationships between EOs. Some of the most complex feature constellations, for example, involve about 15 EO classes plus about the same number of EOR classes. Furthermore, the models can be filtered effectively by relation types that are of interest in a given context. The highly structured and targeted information handling inside the models, which is supported by a user interface displaying objects and their inter-relations graphically, further intuitive and clear model management. Preferably, provisions for recognizing loops and warning the users have to be implemented into the system. As very long chains of automatic EO instantiations may be difficult to be handled, one solution is to define a limit for the length of the chains. All perpetuating links will then be checked later in subsequent ProSAps.

UMEO and MTRT contain *information on class or meta class level*, respectively. In order to expedite their global accessibility and comprehensibility for any ProSAp, it is prefered to represent them in the form of ASCII text using XML syntax and to store them physically inside several tables within a data base. The table structure is kept quite simple and therefore general enough to permit future changes to UMEO. Each class is stored as an XML text entity within a single table cell. The process step applications access the database using standard tools such as ODBC and can navigate through the class models and request the desired information using SQL statements. Open Database Connectivity (ODBC) is a standard interface for accessing relational information stored in data sources of different physical formats. The Standard Query Language (SQL) is a quasi standard for accessing relational databases. Although different SQL dialects exist there is a central set of statements implemented in most versions. The prefered design decisions support the global character of UMEO and MTRT contents. An alternative approach is, for example, to formulize a functional or object-oriented interface in conjunction with one or more application programming interfaces (APIs). ProSAps use API functionality to access the model contents. Compared to the chosen one, this method results in more effort for developing and maintaining the API. On the other hand, it hides the physical model storage, offers he opportunity to implement some high-level retrieval unctionality such as browsing the trees, and takes the proportional amount of implementation effort out of the eature-based applications. However, because ODBC and SQL, and other similar methods also provide some kind of abstraction from database system peculiarities, and because browsing the models does not seem to be too complicated, an advantage of the first alternative is that it relies on quasi standards. *Instance level information*, on the other hand, is stored in two ways: since EO instances are the building blocks of product models, they are stored within the corresponding ProSAp-specific files. EOR instances represent inter-relations between EO instances inside any product model and are therefore preferably stored outside of the product model files, within a dedicated database table.

In order to allow the creation of individual user interfaces which can be personalized to every user, the human machine interface (HMI) has to be treated separately from the internal re[illegible]tation of EO(R) classes. EORs are also a means of f[illegible]lating internal and external EO models, which means th[illegible] inter-relations can be adjusted dynamically according to the user's needs. These adjustments may also be the results of the system's internal knowledge processing.

ULEO supports cooperation implicitly (through EORs) and explicitly (experience transport functionality), thus giving access to a wide variety of information originating from various applications along the process chain. Informational EORs link EO classes and instances logically in any variation. AutoCreate classes are suited for describing actions to be performed in certain situations. How can they be exploited to promote an information flow between applications? The following description is generally valid for all applications.

One option assigns tasks to CAx systems and to an external software module fulfilling basic workflow management functionality. This will, in the following, be designated as the *cooperation service* (*CS*). The set of workflow-specific tasks may vary and can be expanded incrementally to deploy the full potential of the workflow management. The CS may be implemented as part of one or more ProSAps or may be an e[illegible]application which communicates via inter-process r[illegible]h the ProSAps.

[illegible] mixed-biased approach utilizes a watchdog functionality within the ProSAps to be triggered when users desire to <u>instantiate</u> a certain EO class into their product models. Triggered in this way, the ProSAp informs the cooperation service. The CS reads UMEO and checks it for relevant AutoCreate contents. If there are none, the ProSAp is told to continue instantiating the selected EO class, also using UMEO's information on class parameters, etc. If there are AC contents however, they will be interpreted and the applications concerned will be told to instantiate certain additional EO classes. More than one application may be involved. Also, relevant EOR instances will be created and stored. If an EO instance is edited or changed or updated (change management), the ProSAp will inform the CS, which checks the related ProSAp product models by reading the EOR instances and the UMEO contents. Then the desired changes will be propagated by the CS from the initiating ProSAp to the others. The same principles are followed if a user wishes to apply comments or other information to EO(R) instances.

As the task of interpreting AC contents may be quite complex, it seems sensible to implement it as an independent application. Alternatively it may be integrated within the ProSAps or the CS. This will be called *inference machine* in the following.

Concurrent Engineering (CE), including the term's interpretation in the sense of simultaneous work, can be facilitated within the ULEO approach by letting engineers work in parallel and having the ProSAp log the individual EO instantiations. One of several ULEO embodiments is set out very briefly below.

If an EO instantiation has an impact on other product models for which the initiating user is not responsible, the CS will send the system's automatic changes as <u>proposals</u> which have to be accepted or rejected with comments by the receiver. Thus, users are not able to directly manipulate other users' models and changes will not be lost. Furthermore, all users can do their work as usual, benefiting from automated product generation. This method of realizing concurrent engineering seems promising, since it offers engineers more freedom as to when-to-do-what, while insuring correct processing of product model changes.

ULEO is applicable within a wide range of process-step applications. Some practical applications are described in the following.
The approach's benefits for automation are quite obvious. As already set out, EO classes are preferably instantiated automatically, triggered by preceding instantiations. For example, a set of machining planning features can be generated for another set of finish-part features; analogously, measuring elements constituting an inspection plan can be generated according to tolerance objects.

Generally speaking, upstream applications can use information from downstream applications to better be able to estimate the results of particular design decisions (*forward view*). In contrast, downstream ProSAps can use the information stemming from upstream ones to broaden their informational context and, thus, make more profound decisions (*backward view*). Any application can reliably react to product changes handed over to it on the EO instance level. Great advantages ar achieved by linking product models on the EO level bi-directionally using EORs, thus enabling engineers to work within any desired partial model of the product and obtain changes propagated by the system. According to the state of the art, cylinder head designers have to work within rough part and machining models, but not inside the finish-part model; changes have to be propagated verbally. This is relevant not only between different partial models of the same product but also between models of a product on the one hand and its tools on the other.

ProSAps are supported by powerful services such as cost estimation, design-to-X, workflow management, and management of experiences and justifications, e. g.. The informational integration of the product development process chain obtained through cross-linked EOs are utilized by *service applications* (*SAs*), which can provide context-relevant nformation to users of the individual ProSAps. An nformational context preferably comprises several omponents:

a) A certain ProSAp-specific product model of a certain degree of maturity from which the service application is called.
b) The task to be performed next by the user.
c) The respective models in other ProSAps which describe the same product from their specific view.

Service applications have access to the desired contextual nformation by following EORs on the instance level. They arget EORs of certain types which are pertinent for the specific ervice and disregard others. Some examples are set out in the ollowing.

The service of *cost estimation* can be supported. Assuming he costs for machining a certain feature, the representation of a pecial hole in a cylinder head must be estimated. This can be ac[illegible] by tracking EORs of the type "is_machined_as", w[illegible]nect the design feature instances with one or more m[illegible]g feature instances stored within another product model created by the machining planning ProSAp. From there, EOR instances lead to collated machining operations, machining resources, and cost information databases. Thus, all the relevant information for cost estimation is directly and instantly accessible. Also, case-based reasoning approaches for cost estimation harmonize with this method.

In the same way, other service applications for design-to-X can be implemented, supporting designers with enough information to optimize their work with respect to various aspects.

Similarly, applications in the so-called *downstream processes* have access to information from finish-part design in order to understand the *reasons for* special *solutions*, enabling them to react correctly to problems which crop up. This often leads to design-change suggestions, which are handed over to the responsible designer by another service application performing workflow functionality (*cooperation service*, see abo[illegible]. This bi-directional flow of information promotes ef[illegible]nd secure change management.

[illegible]e technology facilitates a more detailed description of p[illegible]cts. In the automotive industry for example, it goes one granularity step below that of the product's component *parts*. EORs allow this feature-based information to be made accessible to *information retrieval* triggered by any information system, such as an Enterprise Resource Planning system, illuminating the current state of product development, e.g. by creating statistic analyses. In this respect, MTRT's hierarchical organization is paramount as it permits the utilization of very specific EORs through knowing the meaning of their more common parent EOR types.

ULEO's EORs preferably support *assembly planning* by providing a means of relating feature instances and/or larger entities such as parts, sub-assemblies, and assemblies to other information such as resources and material. Moreover, EORs are a means of integrating *simulation applications* into the development process chain by inter-relating EOs and simulation objects – provided that the information model is object-oriented and accessible from within other applications.

UMEO preferably contain any amount of *ontological knowledge*, i.e. information describing the meaning of object classes, e.g. by relating them to other object classes using various relationships (EORs). This knowledge can go beyond the inner core set of EO classes necessary, for example, for feature-based engineering. By continually adding ontological knowledge, UMEO slips into the role of an *ontological knowledge base*. The same result is achieved by coupling the core EO(R) classes to a separate ontology using EORs. In both cases the ontological knowledge is preferably made accessible to the users of the ProSAps, helping them to fulfill their tasks. Examples are information on when to use exactly which feature classes or a collection of problems often occurring in the context of an EO class. This may include knowledge and information about best practices.

UMEO provides the technology for *handling users' experiences* very efficiently. A kind of *knowledge management service application* can allow users to attach information to any EO and EOR instance and offers appropriate means of retrieving it again. Attaching experience objects to feature instances, for example, is useful to record the reason for choosing just that feature in precisely that context. Attaching it to EOR instances may document the reason for the use of machining feature set B for the manufacturing of a set A of design features. Another key field of application for experience handling is *change management*. Changes applied to an EO instance by user U1 in model M1 may be given to another user U2 who is responsible for the model M2 and whose EO instances have to be adjusted to the changes in M1. EORs between EO instances mark the path between EO instances, models, and users. Attaching and maintaining *user models* can be of great help to engineers while fulfilling their task. So-called user stereotypes are one of the first and probably also one of the most straight-forward approaches. *Uncertainty*, which is frequently inherent to user models, is often handled by using methods of *uncertain reasoning*.

Preferably an implementation of ULEO uses the base functionality of a commercial CAD system and extend it by having additional functionality within the discussed scope.

The automotive cylinder head application example cover detail design of cylinder heads and mold tools, together with the corresponding machining planning. One implementation variant is deployed to gain quick and general experiences based on a restricted feature constellation linking functionality. It is therefore being implemented using the Basic programming language. A further variant offers the general feature linking functionality together with a good user interface and is coded using C++. Details are set out in the next two sections.

The Feature Constellation Linking (FCL) application implements the instantiation, editing, and deletion of the feature constellations '*cylinder head/crankcase bolting*' and '*hole with boss*'. A simple annotation service is included, permitting users to add annotations (e.g. experiences or substantiations) to each and every element of the feature constellations.

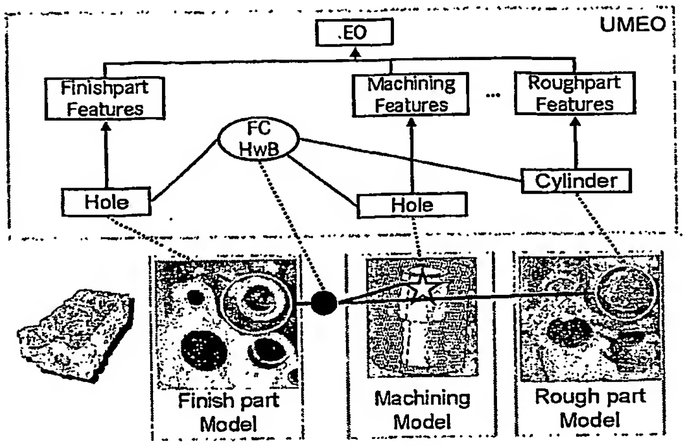


Figure 1: Feature Constellation 'Hole with Boss'

[illegible]ils of the application process: The bolting FC allows to [illegible]tiate a bolted connection between cylinder head and crankcase in one action after creating the reference elements. Having informed the user about the automatic insertions initiated, the Basic macro creates holes through both finish parts (two different CAD models) and instantiates the bolt within the assembly model. Synchronization of all the elements' parameter values is maintained. The EOR instances are stored within a relational database. After the instantiation is finished, the user of any of the instances in any model can be selected for being edited and for changing and updating its parameters. The system will generate information about the effects of the changes made on other EO instances and synchronize their parameter values. The information about the related FC elements is read from the EOR entries inside the Excel table. In a first version of the implementation, the features are hard-coded within the macros. In the enhanced version, the information on EO and FC classes is stored in XML-formatted text files and interpreted during runtime. The an[illegible]n functionality is offered for each FC element. By p[illegible] button within the edit dialog, the user can activate an an[illegible]n window for textual comments or messages. If any of the other models are opened, the user is informed about the new annotation on some EO instance. Users can read the annotation text by clicking the respective button. The 'hole with boss' feature constellation, as shown in figure 1, works according to the same principles and lets the user insert three *hole* feature instances into three different part models in one action: finish-part feature, rough-part feature and machining-feature instance.

The General Feature Linking (GFL) application is able to perform inferences on AutoCreate classes. It supports the same process steps as the FCL application plus the so-called '*ejector*' and '*inspection*' functionalities. EOR instances are stored in a relational database. As the system is able to interpret AutoCreate classes, it can perform automatic instantiations. Three different alternatives for inference machines can be integrated. One is an enhanced add-on to the CAD system's knowledge processing component, another is a software implementation of the information manager known from D. Lutters:"Manufacturing Integration Based on Information Management", Ph.D. thesis, University of Twente, Enschede, 2001. and the third is the ontology-based knowledge modeling system 'Ontoworks', developed in a collaborative effort between the University of Stanford and the automotive industry. The first method is internal to the CAD system whereas the other two are implemented as separate software applications that communicate with the CAD application. In addition to the broader functionality of the first method, it has an enhanced user interface due to the software integration into the CAD system, which is achieved using the respective application development environment.

Details of the application 'ejector': The feature constellation '*ejector*' covers features and parts to be used for the following product (partial) models: cylinder head finish part and rough part, mold-&-die tool finish part, and machining. To be exact, it instantiates an ejector pin and its corresponding holes in the mold tools, as well as the contact bodies as part of the rough cylinder head, and the machining features inside the cylinder head's machining model.

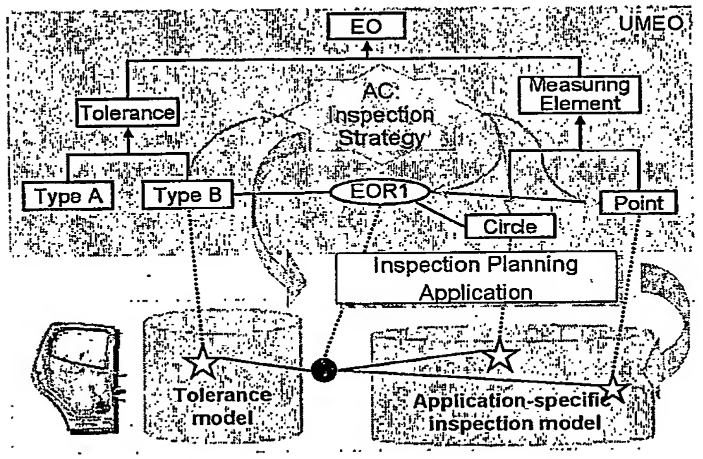
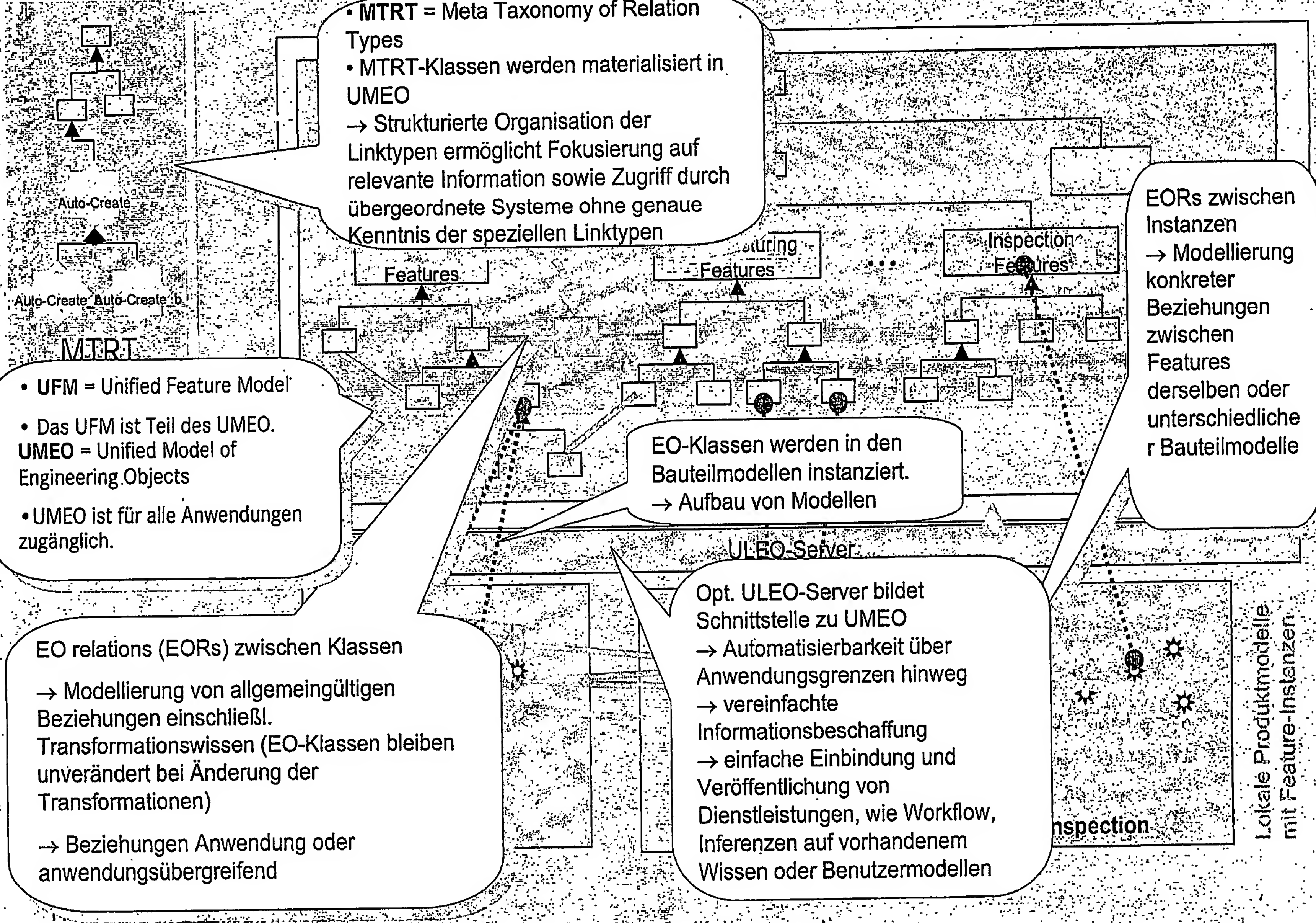


Figure 2: Application 'Inspection'

Details of the application 'inspection': This application covers aspects of finish-part design, tolerancing, and inspection planning for body in white. Preferably, sheet metal parts are toleranced using dedicated tolerance objects from UMEO. The tolerances are put together according to inspection tasks and automatically converted into appropriate measuring elements, which constitute the building elements of the inspection program. These benefits of automation are coupled with process chain integration, in that, for example, measurement results are to be evaluated and routed back to the part's nominal geometry. Automatic deduction of measuring programs is achieved by the following methodology: AutoCreate classes within UMEO store inspection planning-neutral rules, so-called *inspection strategies*, which are interpreted by an inspection planning application (IPA) as depicted in figure 2. During the interpretation process, the IPA generates measuring elements, lescribing the processing of the measurement results and EOR ıstances. Figure 2 shows the EO(R) classes and instances ıvolved. The measuring results are passed to an analysis ystem by the coordinate measuring machine and stored within ι specific file. Using the EORs instances, the measurement esults is preferably routed back to the tolerance and finish-part eature instances. Thus, if quality problems are detected, ınnotations are preferably generated by the inspection planner ınd transported to the designer's computer for generating nformation about them at the granularity level of single feature nstances. The designer will know exactly which feature nstances have led to quality problems. Body-in-white feature constellations will be implemented in a version of the system in order to let the designer simultaneously instantiate more than one feature such as flanges or beads together with tolerances into several product models such as class-A car exterior models and car interior models.

DaimlerChrysler Forschung & Technologie

UMEO - alle Features und andere relevante Klassen in einem Modell



• **MTRT** = Meta Taxonomy of Relation Types
• MTRT-Klassen werden materialisiert in UMEO
→ Strukturierte Organisation der Linktypen ermöglicht Fokusierung auf relevante Information sowie Zugriff durch übergeordnete Systeme ohne genaue Kenntnis der speziellen Linktypen

• **UFM** = Unified Feature Model
• Das UFM ist Teil des UMEO. **UMEO** = Unified Model of Engineering.Objects
• UMEO ist für alle Anwendungen zugänglich.

EO-Klassen werden in den Bauteilmodellen instanziert.
→ Aufbau von Modellen

EORs zwischen Instanzen
→ Modellierung konkreter Beziehungen zwischen Features derselben oder unterschiedlicher Bauteilmodelle

EO relations (EORs) zwischen Klassen

→ Modellierung von allgemeingültigen Beziehungen einschließl. Transformationswissen (EO-Klassen bleiben unverändert bei Änderung der Transformationen)

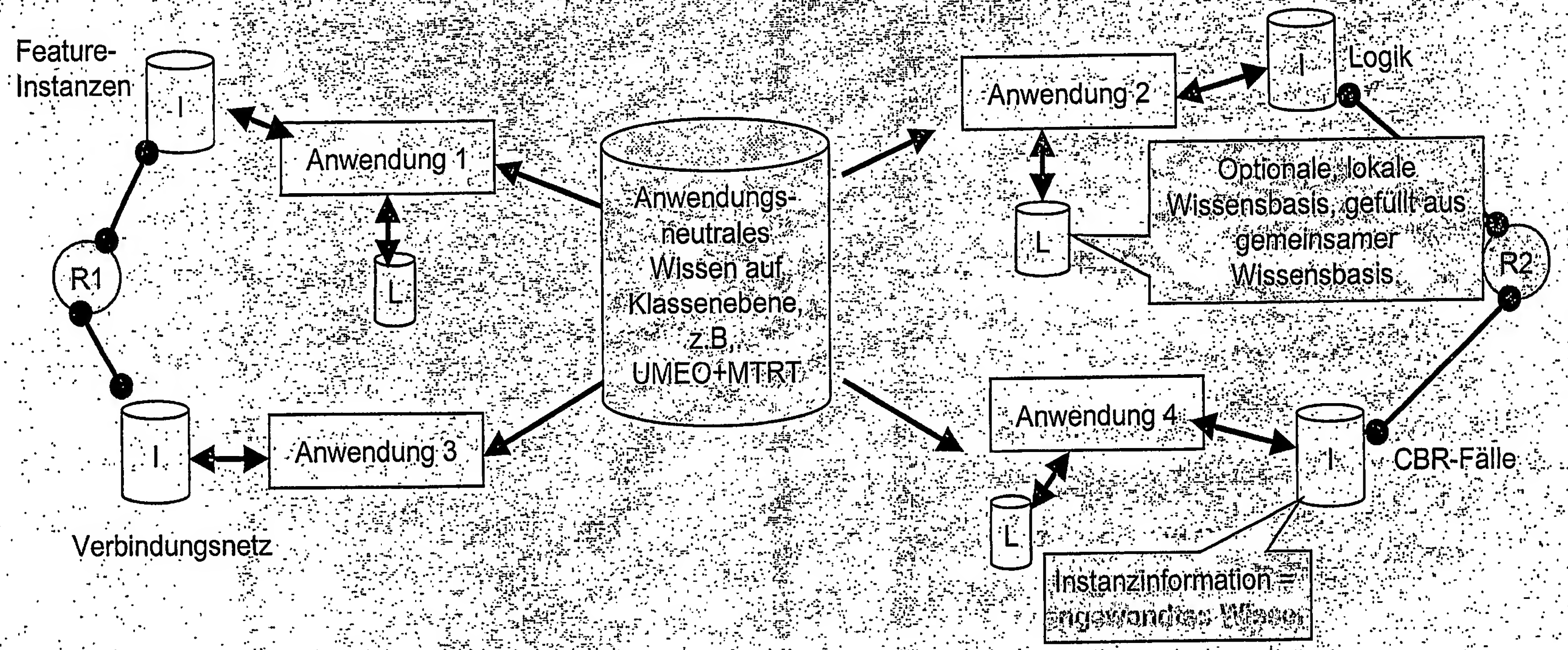
→ Beziehungen Anwendung oder anwendungsübergreifend

Opt. ULEO-Server bildet Schnittstelle zu UMEO
→ Automatisierbarkeit über Anwendungsgrenzen hinweg
→ vereinfachte Informationsbeschaffung
→ einfache Einbindung und Veröffentlichung von Dienstleistungen, wie Workflow, Inferenzen auf vorhandenem Wissen oder Benutzermodellen

DAIMLERCHRYSLER

Forschung & Technologie

Integration von Anwendungen entlang der Prozeßkette...

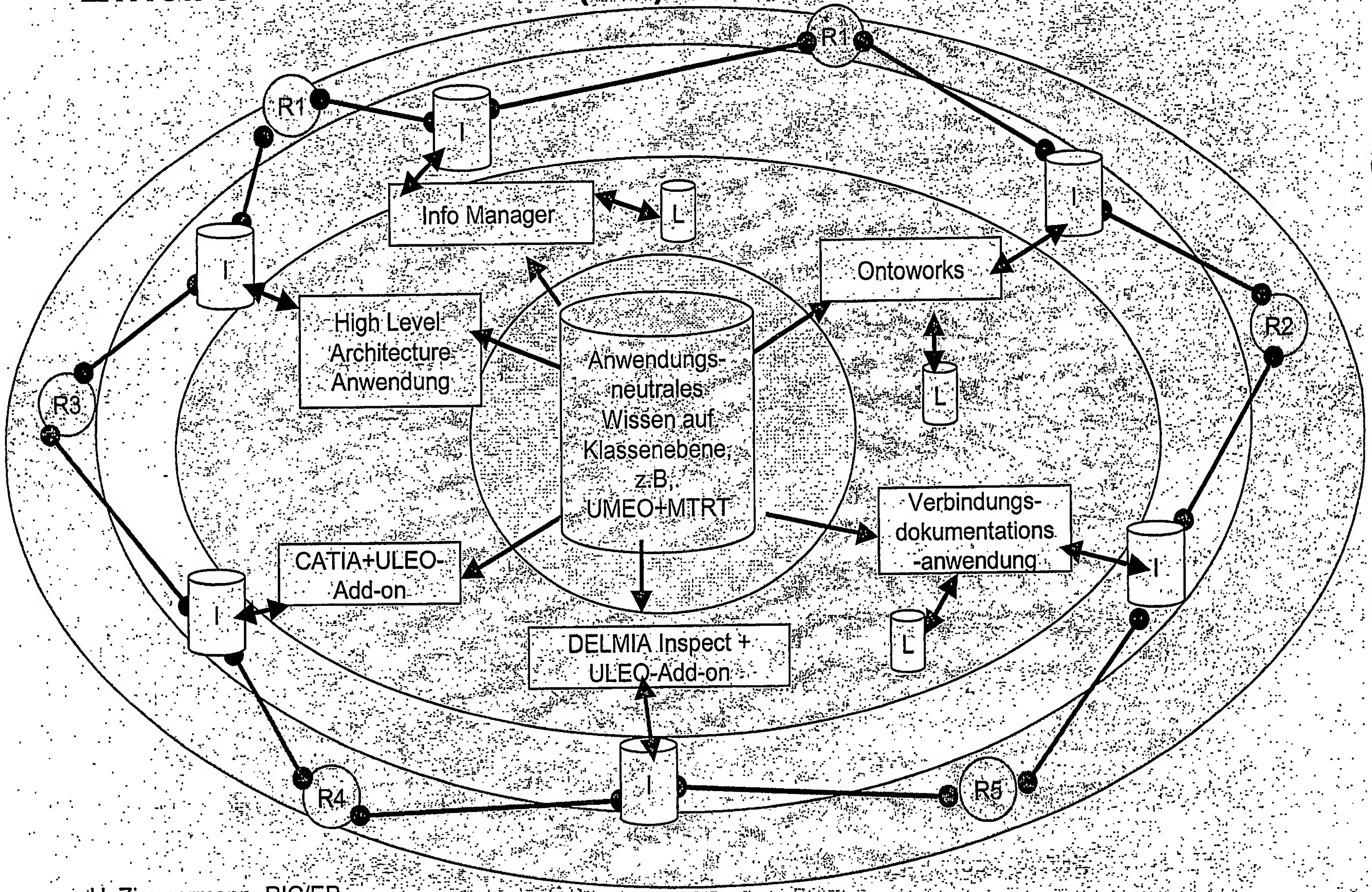

Feature-Instanzen
I
Anwendung 1
L
R1
I
Anwendung 3
Verbindungsnetz
Anwendungs-neutrales Wissen auf Klassenebene, z.B. UMEO+MTRT
Anwendung 2
I
Logik
L
Optionale lokale Wissensbasis, gefüllt aus gemeinsamer Wissensbasis
R2
Anwendung 4
I
CBR-Fälle
L
Instanzinformation = angewandtes Wissen

H. Zimmermann, RIC/EP

DAIMLERCHRYSLER

Forschung & Technologie

Zwiebelschalenmodell (1/4)...

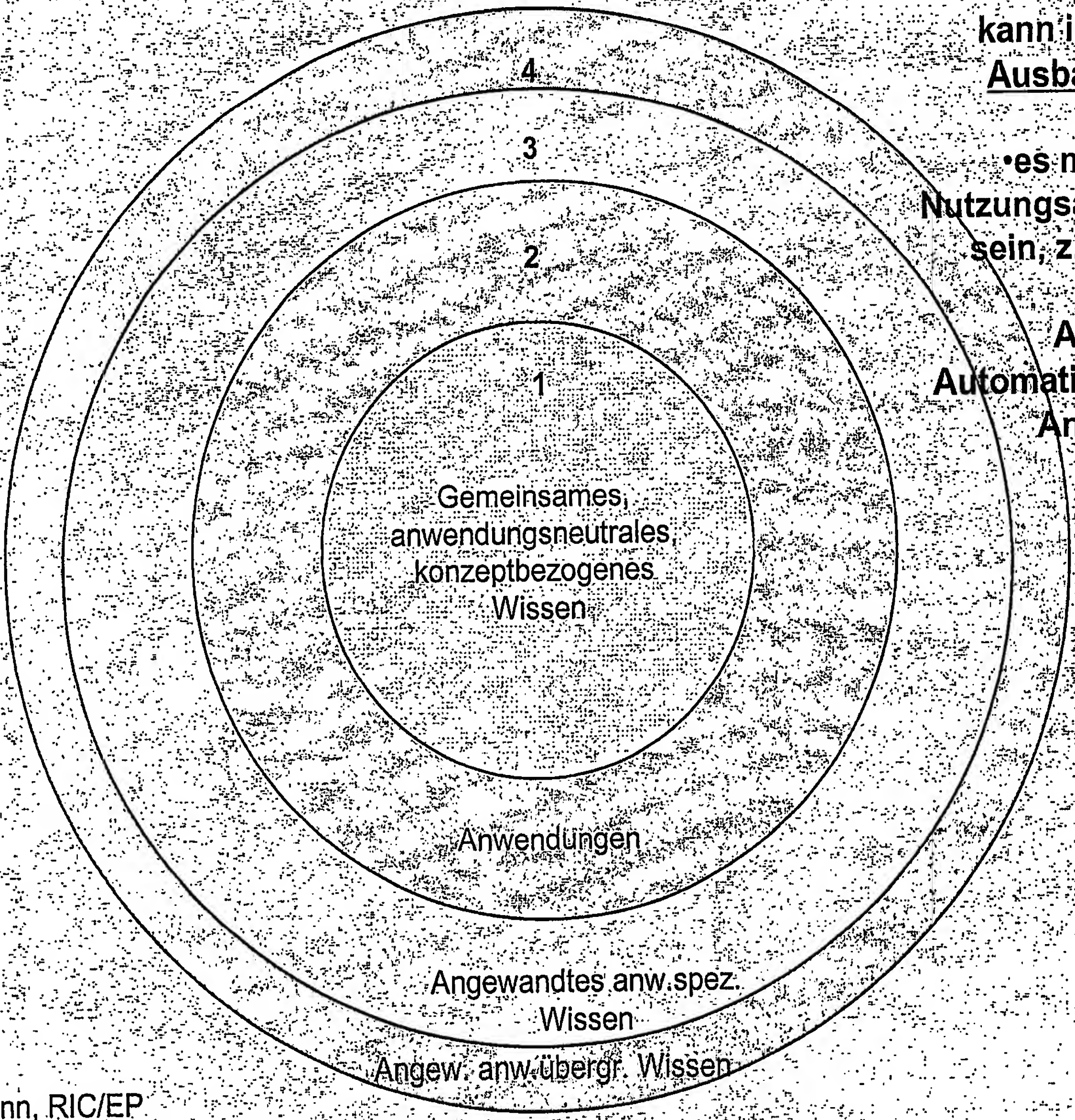

R1
R1
I
Info Manager
L
I
Ontoworks
R2
I
High Level Architecture Anwendung
L
R3
Anwendungs-neutrales Wissen auf Klassenebene, z.B. UMEO+MTRT
Verbindungs-dokumentations-anwendung
I
CATIA+ULEO-Add-on
I
DELMIA Inspect + ULEO-Add-on
L
R4
I
R5

H. Zimmermann, RIC/EP

DAIMLERCHRYSLER

Forschung & Technologie

Zwiebelschalenmodell (2/4)...


4
3
2
1
Gemeinsames, anwendungsneutrales, konzeptbezogenes Wissen
Anwendungen
Angewandtes anw.spez. Wissen
Angew. anw.übergr. Wissen

- Gemeinsames Wissen kann in verschiedenen Ausbaustufen genutzt werden.
- es muß nicht für alle Nutzungsarten vollständig sein, z.B. sehr einfache feature-basierte Anwendungen bis Automatisierung (FL) und Anwenderberatung

H. Zimmermann, RIC/EP

Zwiebelschalenmodell (3/4)

- Hoher **Integrationseffekt** für verschiedene Anwendungen, jede Anwendung kann ihre Stärken voll ausspielen. Gleichzeitig kann jede Anwendung vollständig „ausgeklinkt" werden.
- **Skalierbarkeit** der Wissensnutzung führt zu sehr vielfältiger **Einsetzbarkeit** und hoher **Rückwärtskompatibilität**
  - z.B. bei Features: früher wenige Feature-Klassen ohne Relationen, künftig viele zusätzliche EO-Klassen und EORs
  - Positiv für schrittweise Übernahme von Prototypen in kommerzielle Systeme mit z.T. eingeschränkten Fähigkeiten, wie CATIA/ENOVIA/SMARAGD etc.
- Eine **offene Repräsentationsform** ist wichtig für die Erhaltung und **Konsistenz** des Wissens und seine anwendungsübergreifende **Zugänglichkeit**.

Zwiebelschalenmodell

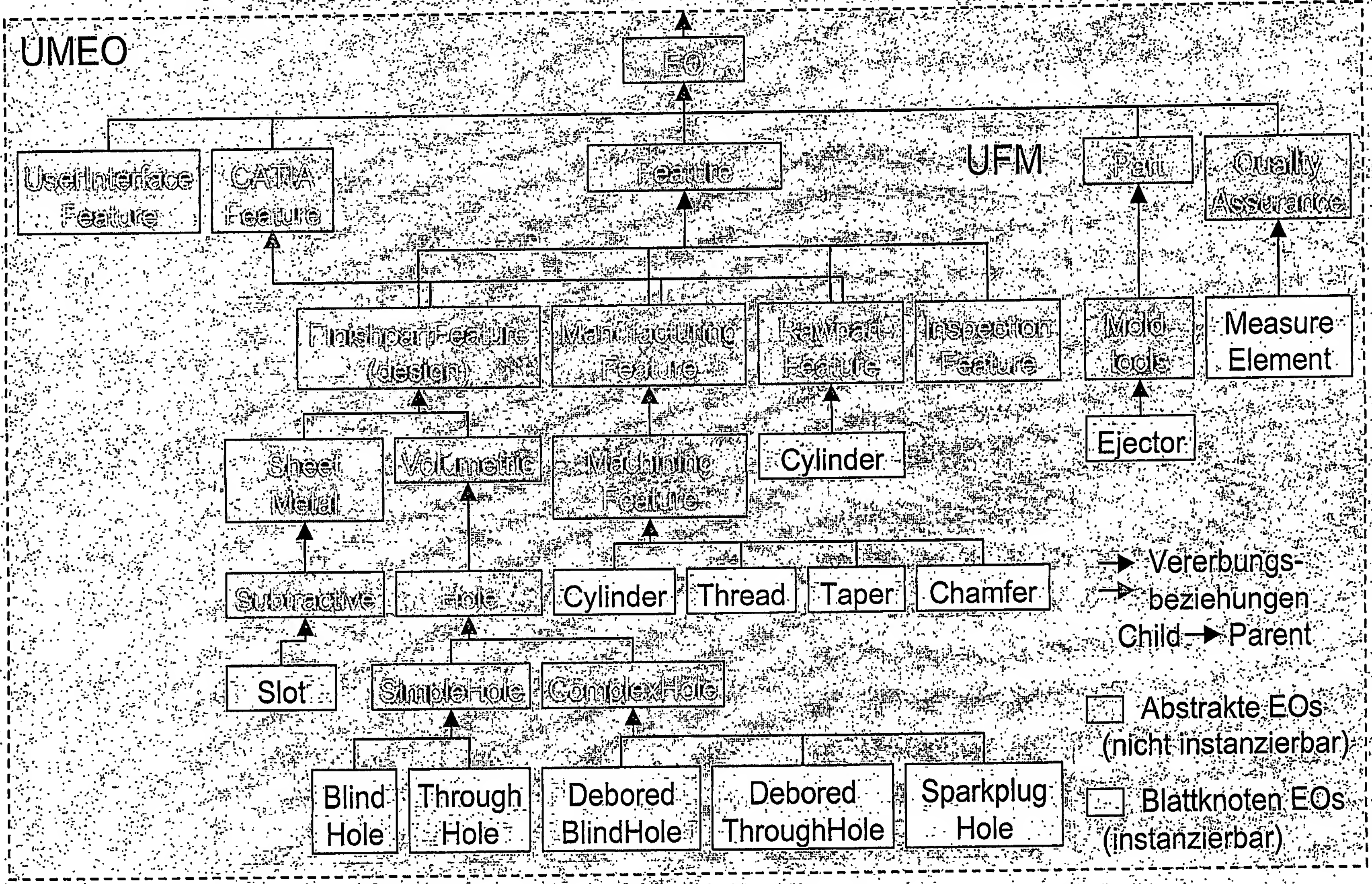
- Integration auf Klassen- und Instanzebene ermöglicht **maximale Kontextinformation** für die einzelnen Anwendungen.
- **Begrenzung** der **Komplexität** nötig (erreichbar durch strukturierte Repräsentation und Filterungsmöglichkeiten, z.B. MTRT)

Zusammenfassung...

- Anwendungsneutrale Wissensbasis (z.B. UMEO/MTRT)
- ...soll von verschiedenen Anwendungen gelesen werden können
- sollte möglichst universelle Modellierung erlauben
- Begrenzung der Komplexität
- Nutzen:
  - Damit werden beliebige Kombinationen zwischen unseren Technologien möglich.
  - Hohe Skalierbarkeit und zugehörige positive Folgen
  - Universelle Zugänglichkeit des Wissens
  - Maximale Kontextinformation

DAIMLERCHRYSLER Forschung & Technologie

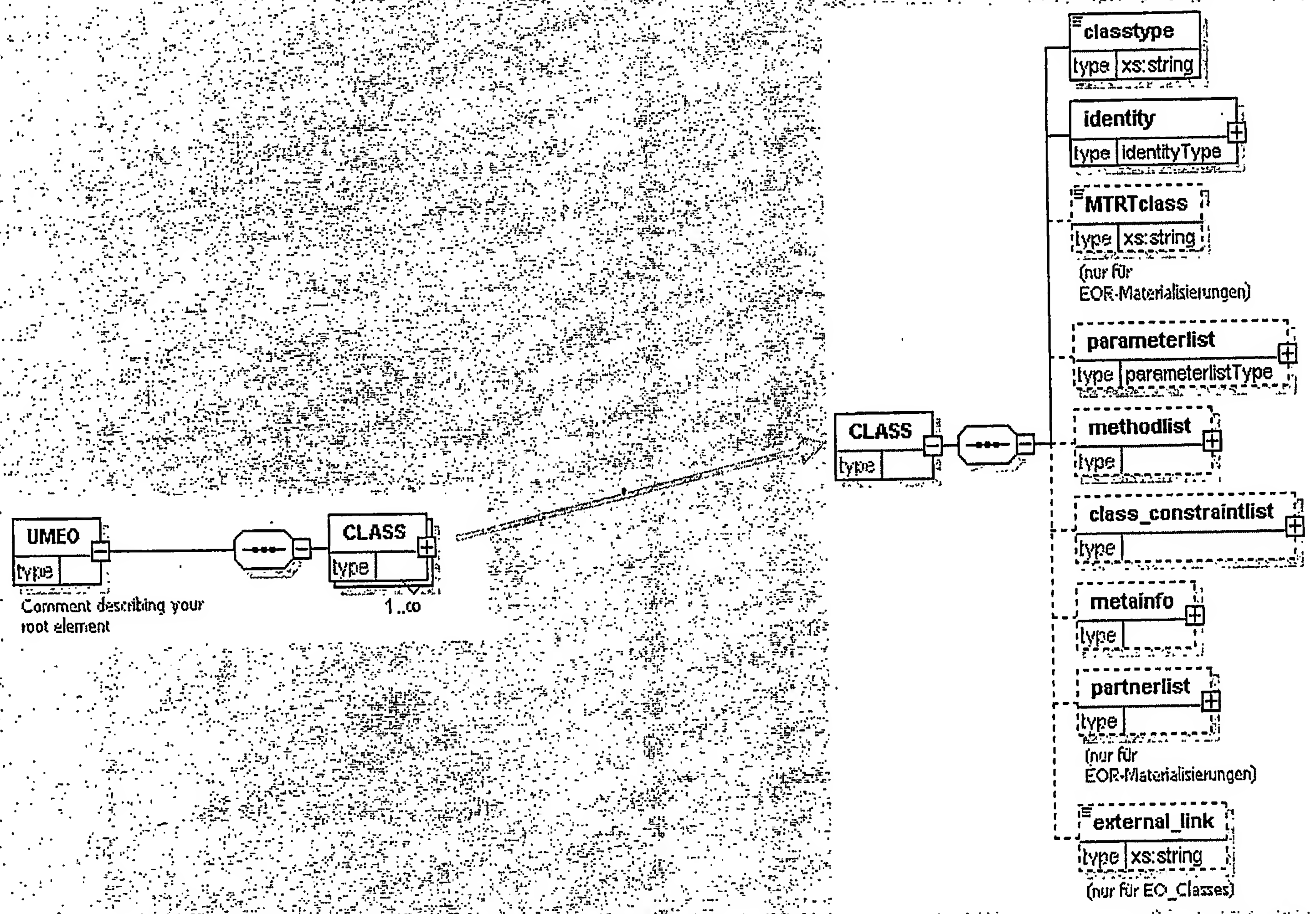
Ausschnitt aus UMEO-Taxonomie



H. Zimmermann, RIC/EP

DAIMLERCHRYSLER Forschung & Technologie

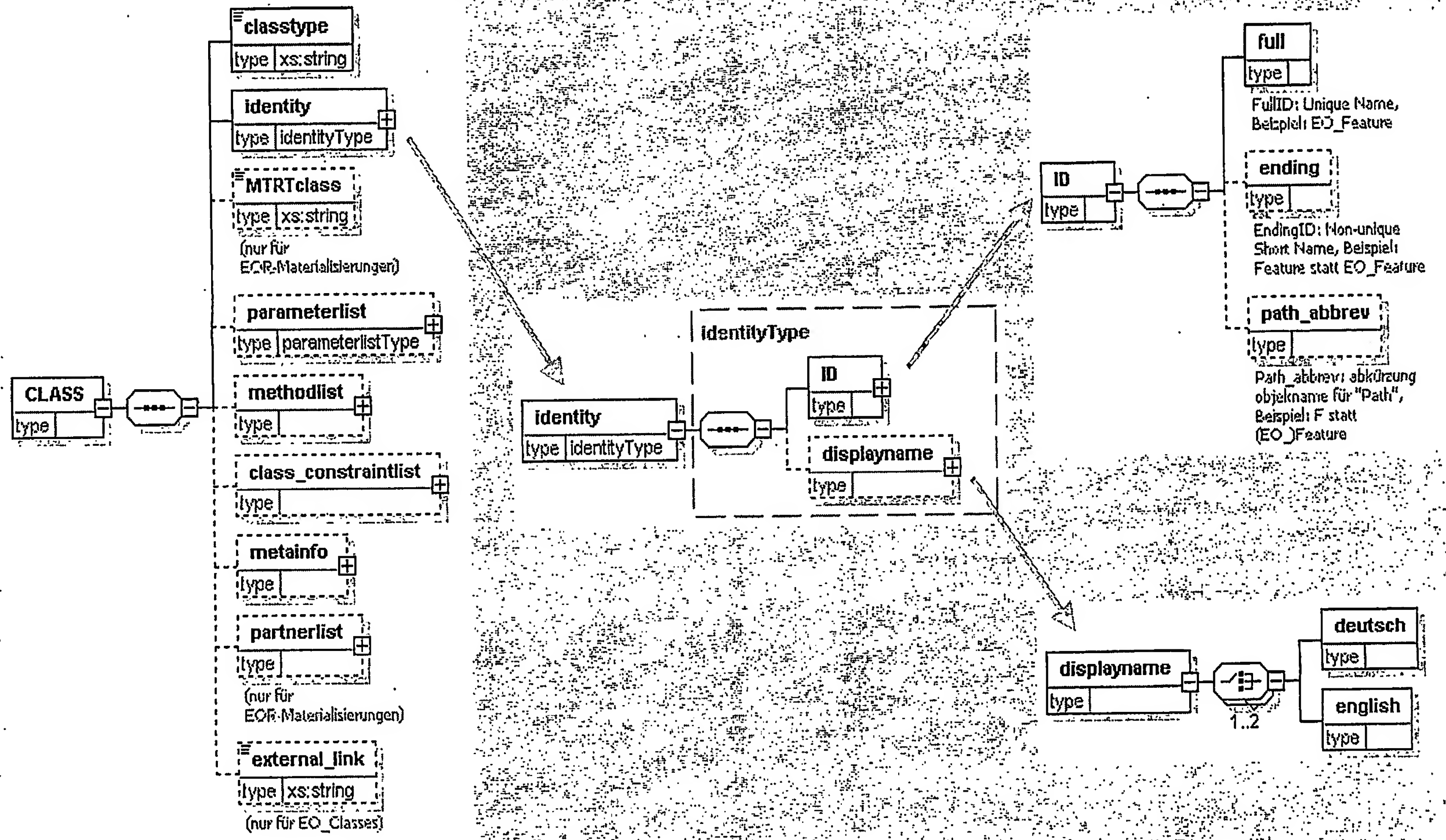
XML-Stuktur zur Speicherung von UMEO



H. Zimmermann, RIC/EP

DAIMLERCHRYSLER Forschung & Technologie

XML-Stuktur (2) - identity

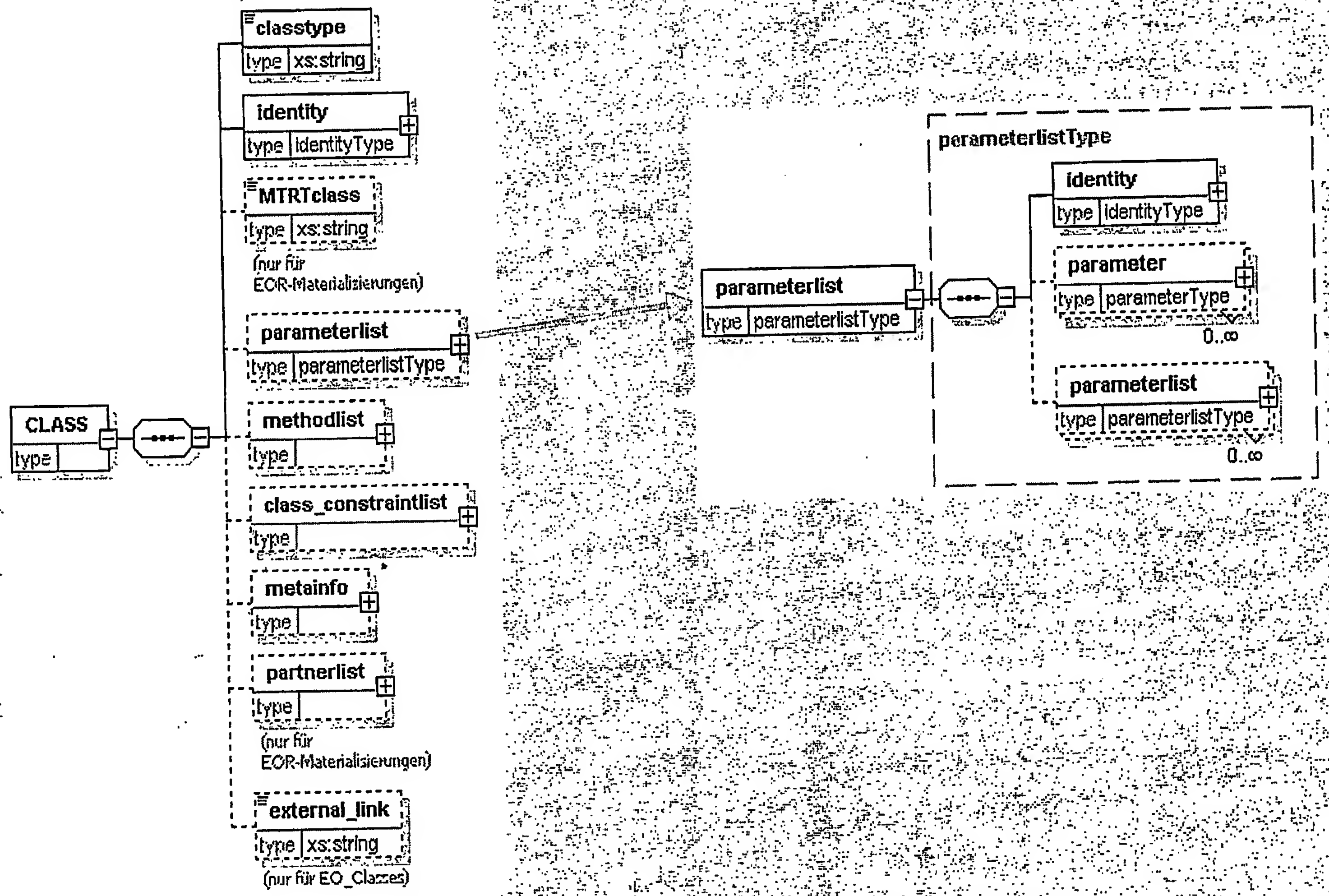


H. Zimmermann, RIC/EP

DAIMLERCHRYSLER

Forschung & Technologie

XML-Stuktur (3) - parameterlist

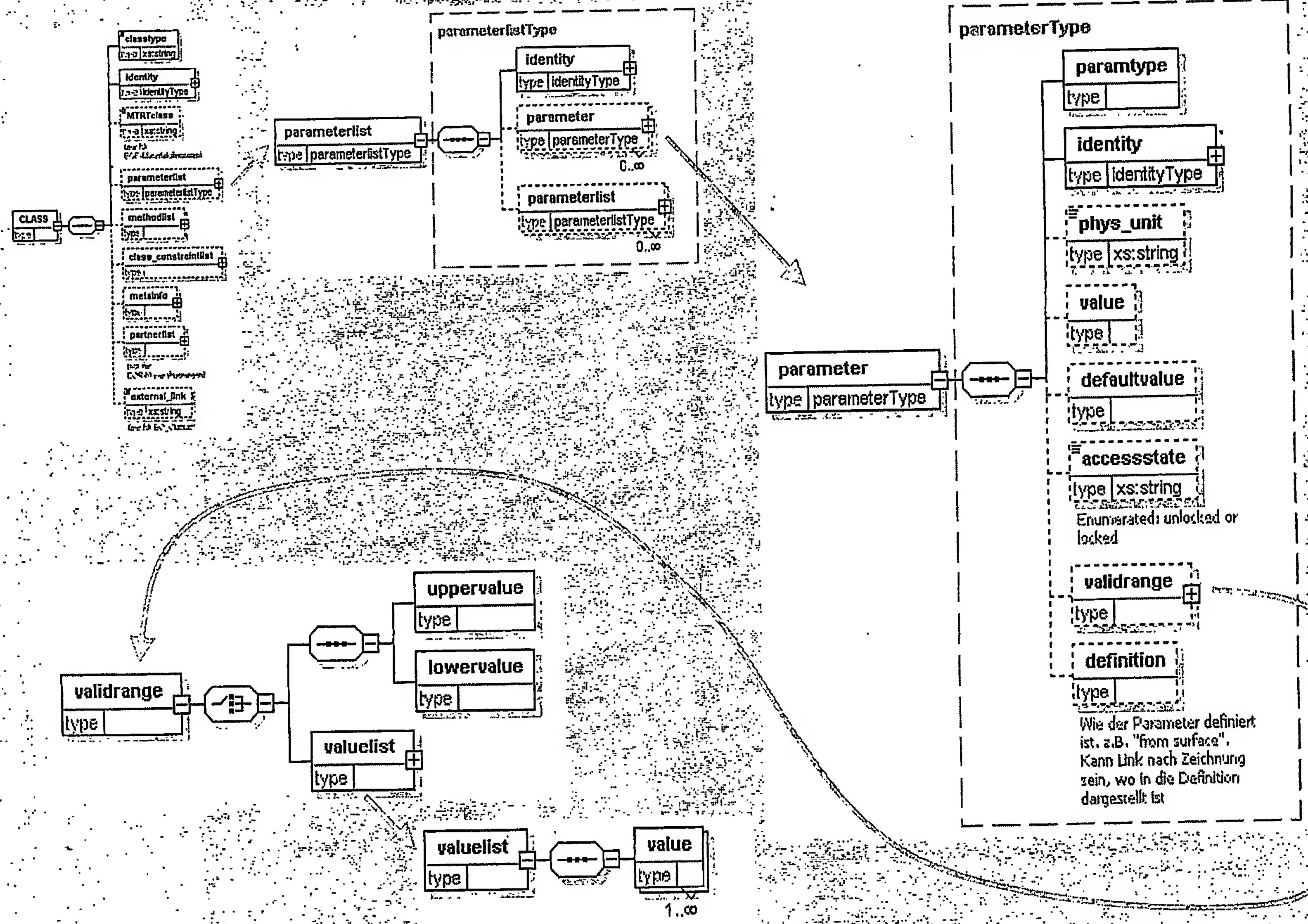


H. Zimmermann, RIC/EP

DAIMLERCHRYSLER

Forschung & Technologie

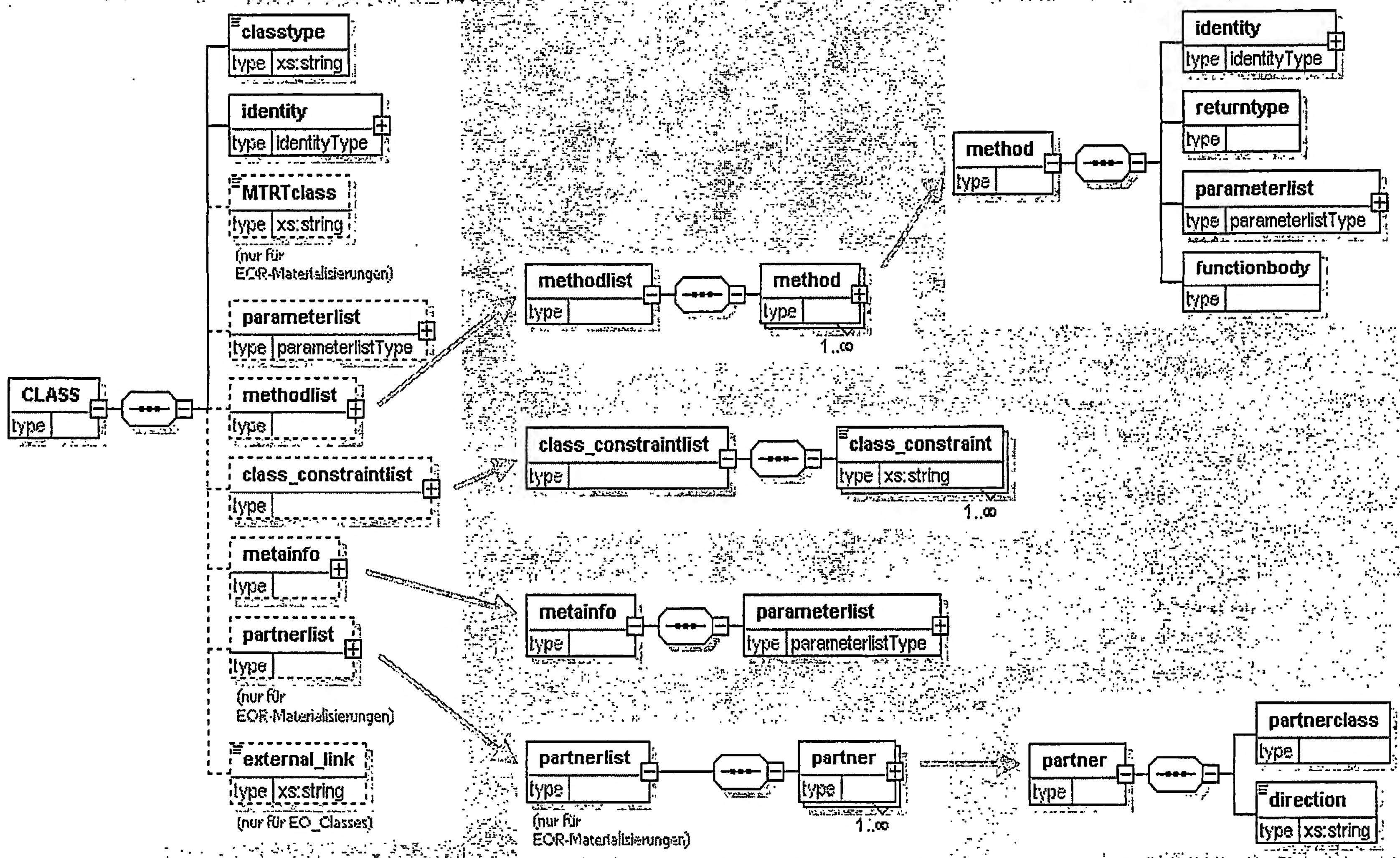
XML-Stuktur (3) - parameterlist.parameter



H. Zimmermann, RIC/EP

DAIMLERCHRYSLER Forschung & Technologie

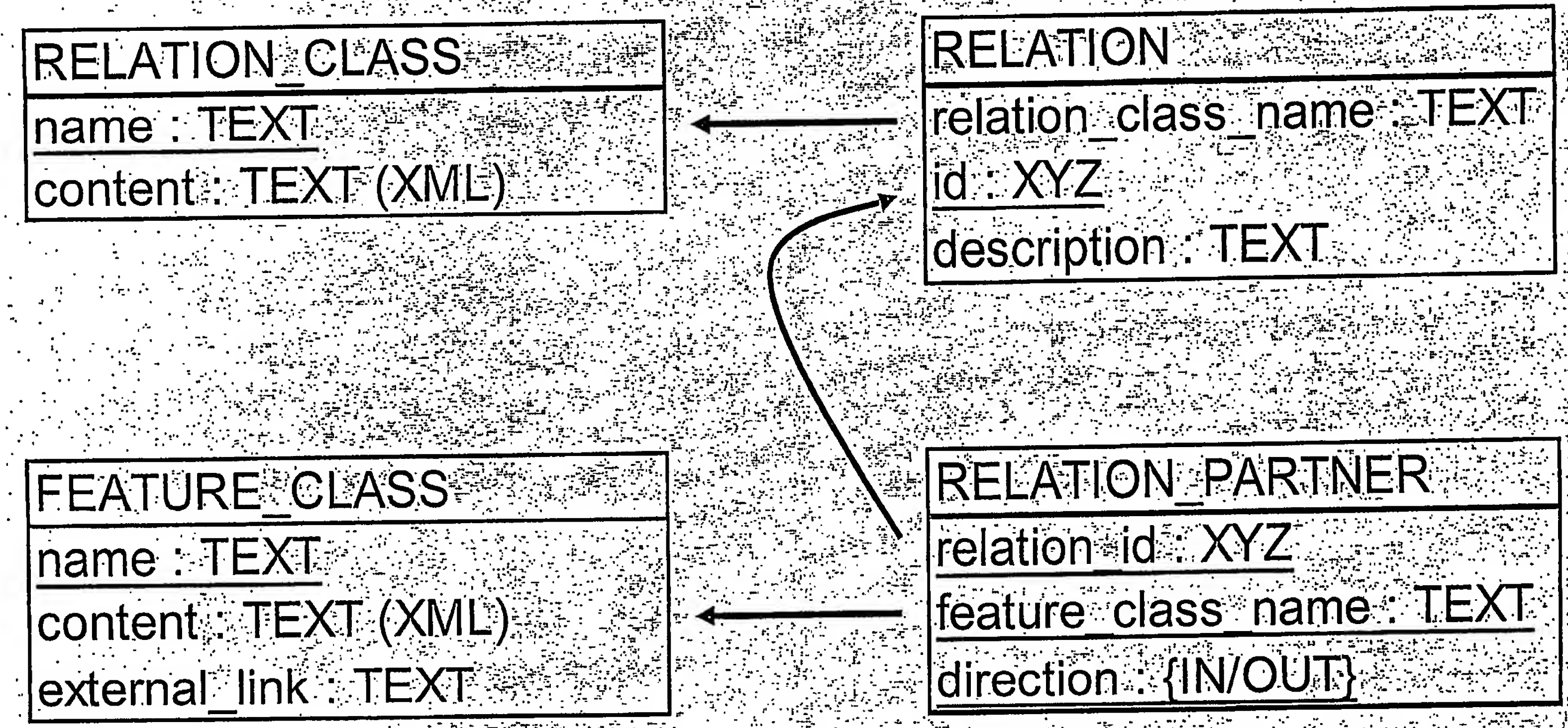
XML-Stuktur (2) - methodlist, class_constraintlist, metainfo and partnerlist



H. Zimmermann, RIC/EP

DAIMLERCHRYSLER Forschung & Technologie

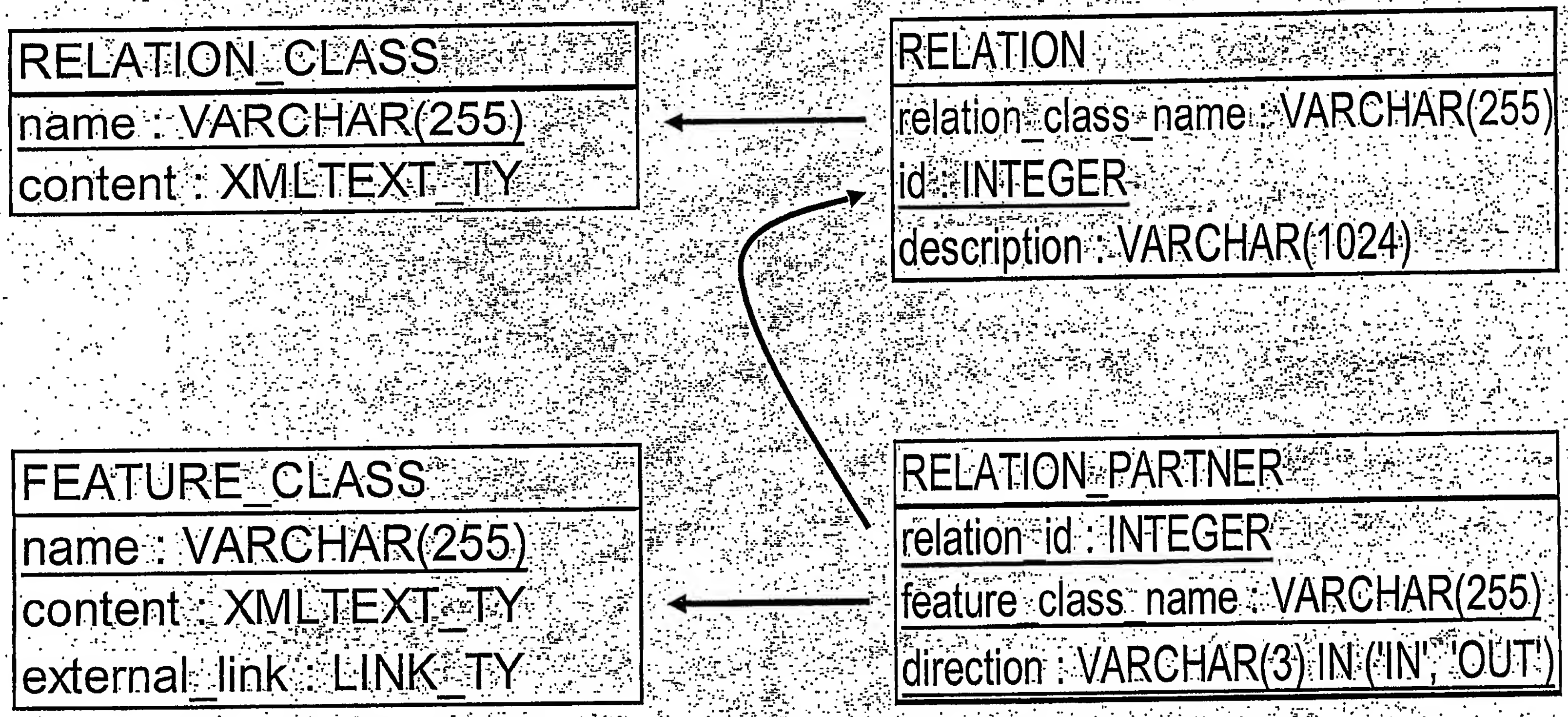
Übersicht Tabellen



Für TEXT (XML) kann man den Datentyp CLOB verwenden. Idealerweise sollte mit DISTINCT TYPE ein eigene Datentyp erzeugt werden.

DAIMLERCHRYSLER Forschung & Technologie

Übersicht Tabellen2



Der Datentyp XMLTEXT_TY entspricht dem Datentyp CLOB mit einer Größe von 100KB.

H. Zimmermann, RIC/EP

Hinweise

- Primärschlüsselspalten können nur maximal 255 Zeichen lang sein.
- Die Primärschlüssel der Tabellen FEATURE_CLASS und RELATION sind Fremdschlüssel in der Tabelle RELATION_PARTNER (ebenso RELATION_CLASS mit RELATION).
- Das Löschen von Datensätzen erfolgt kaskadierend, falls ein Schlüssel in einem anderen Datensatz als Fremdschlüssel auftaucht. (D.h. evtl. werden zusätzlich Datensätze aus der Tabelle RELATION_PARTNER gelöscht.)
- Die Werte der Tabellen sind case sensitive.

Konsistenzwahrung

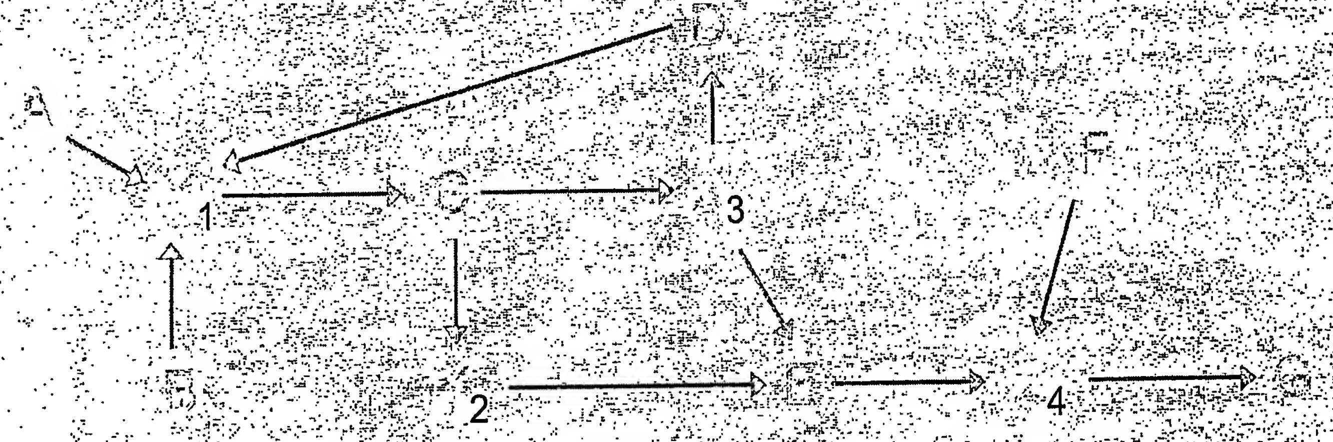
- Kein direktes Schreiben/Ändern auf den Tabellen RELATION und RELATION_PARTNER, da sonst keine Konsistenz gewährleistet werden kann

- Das konsistente Schreiben/Ändern könnte über eine Stored Procedure gelöst werden

- möglicher Prozedurkopf:

```
insert_relation( relation_class_name,
                 description,
                 LIST_OF(feature_class_name_in),
                 LIST_OF(feature_class_name_out))
```

Beispiel



Legende:

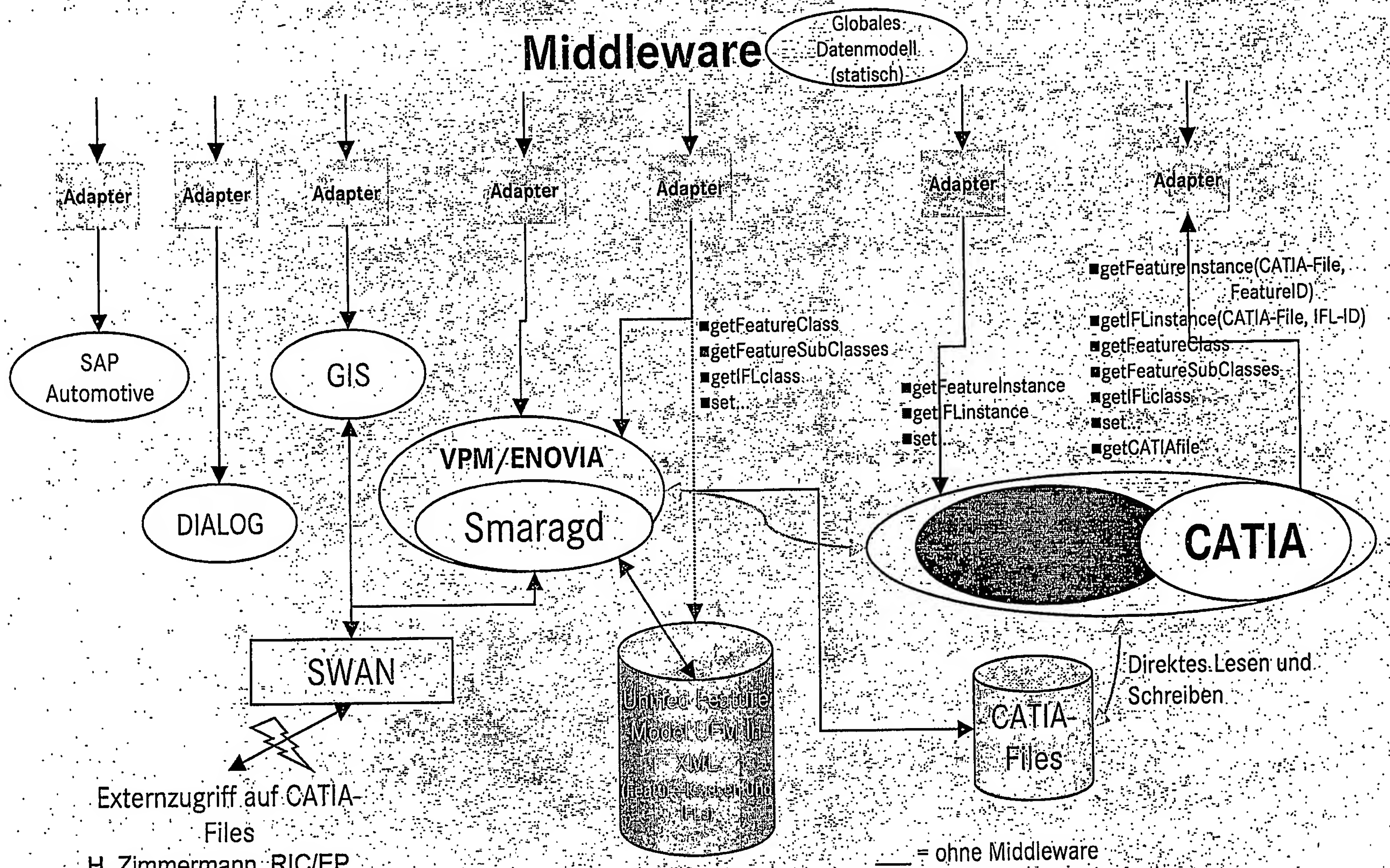
RELATION_PARTNER

FEATURE_CLASS

RELATION

DAIMLERCHRYSLER Forschung & Technologie

Feature-Modellierung und Middleware

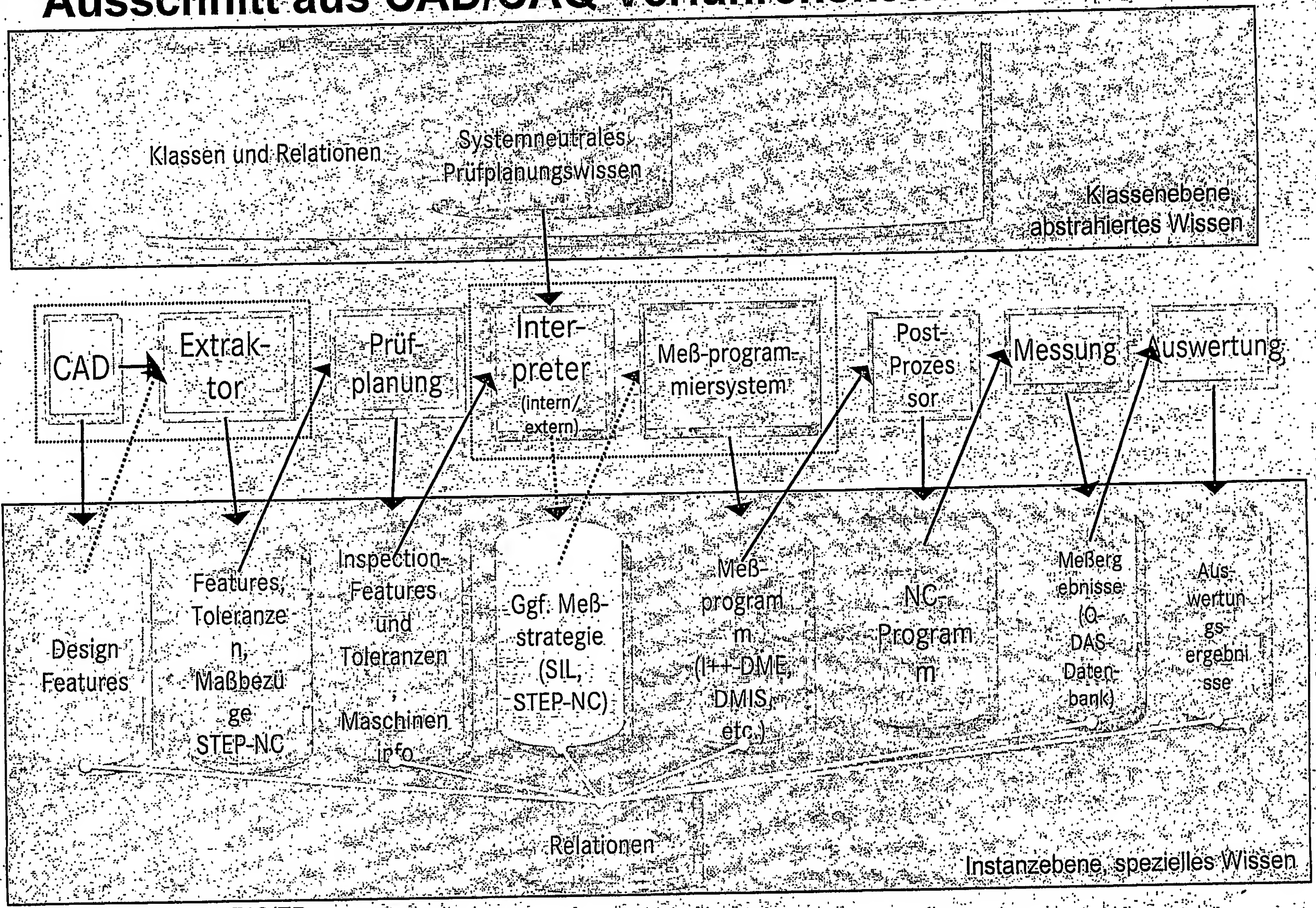


H. Zimmermann, RIC/EP

DAIMLERCHRYSLER

Forschung & Technologie

Ausschnitt aus CAD/CAQ-Verfahrenskette

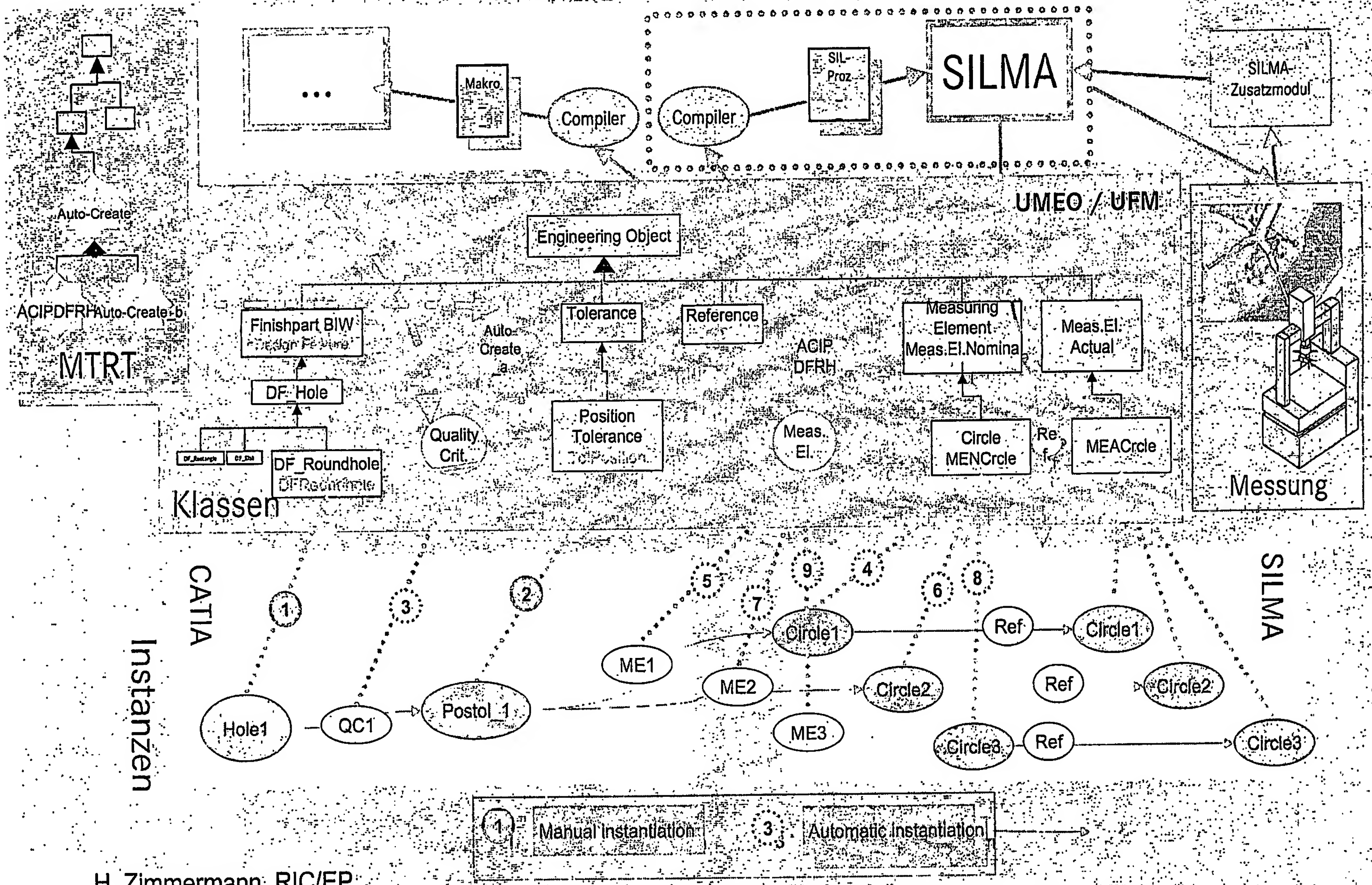

Klassen und Relationen
Systemneutrales Prüfplanungswissen
Klassenebene, abstrahiertes Wissen
CAD
Extrak-tor
Prüf-planung
Inter-preter (intern/ extern)
Meß-program-miersystem
Post-Prozes sor
Messung
Auswertung
Design Features
Features, Toleranzen, Maßbezüge STEP-NC
Inspection-Features und Toleranzen, Maschinen info
Ggf. Meß-strategie (SIL, STEP-NC)
Meß-programm (I++-DME, DMIS, etc.)
NC-Programm
Meßergebnisse (Q-DAS-Datenbank)
Aus-wertungs-ergebnisse
Relationen
Instanzebene, spezielles Wissen

H. Zimmermann, RIC/EP

DAIMLERCHRYSLER

Forschung & Technologie

CAD/CAQ - Klassen, Instanzen, Relationen, Informationsflüsse

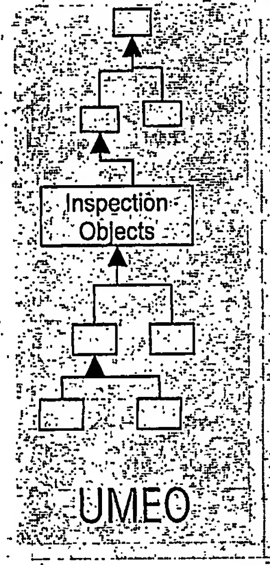


H. Zimmermann, RIC/EP

DAIMLERCHRYSLER

Forschung & Technologie

Beispiel: Meßplanung für ein Loch (C++ Syntax)



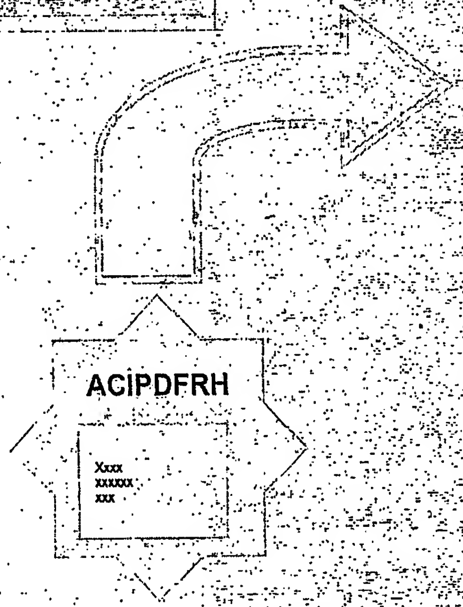
Objekte und Methoden

```
int CreateMeasuringPlan(DF_Hole &oDFhole)
{
        // Tasterdefinition...
        KugelTaster K02A1K13_1(   INDEX, POL, 45.0000, 90.0000, 0.0000, 0.0000
                                  1.0000, 471.8321, 3.0000);

        // Tasterselektion...
        K02A1K13_1.Select();

        // Feature-Definition...
        FeatureCircle REFCIRCNL(  INNER, CART, 186.2358, 155.9298, 17.7250,
                                  -0.0000, -0.7660, 0.6428, 9.9869);

        // Meßdefinition
        ME_Circle *pMEcircle1 = new ME_Circle(); /* "new" erzeugt dauerhafte
                                                     Objekte (Instanz) */
        oDFhole.EOLink(pMEcircle, MeasEl); // verbinde Design-Feature und Meßelement
        pMEcircle1->Method = tactile;
        if (oDFhole.Diameter < 7 then
        {
                pMEcircle1.Adapter = true;
                pMEcircle1. No_of_points = 4;
        }
        else
        {
                pMEcircle1.Adapter = false;
                pMEcircle1.No_of_points = 5;
        }
} // end method CreateMeasuringPlan
```

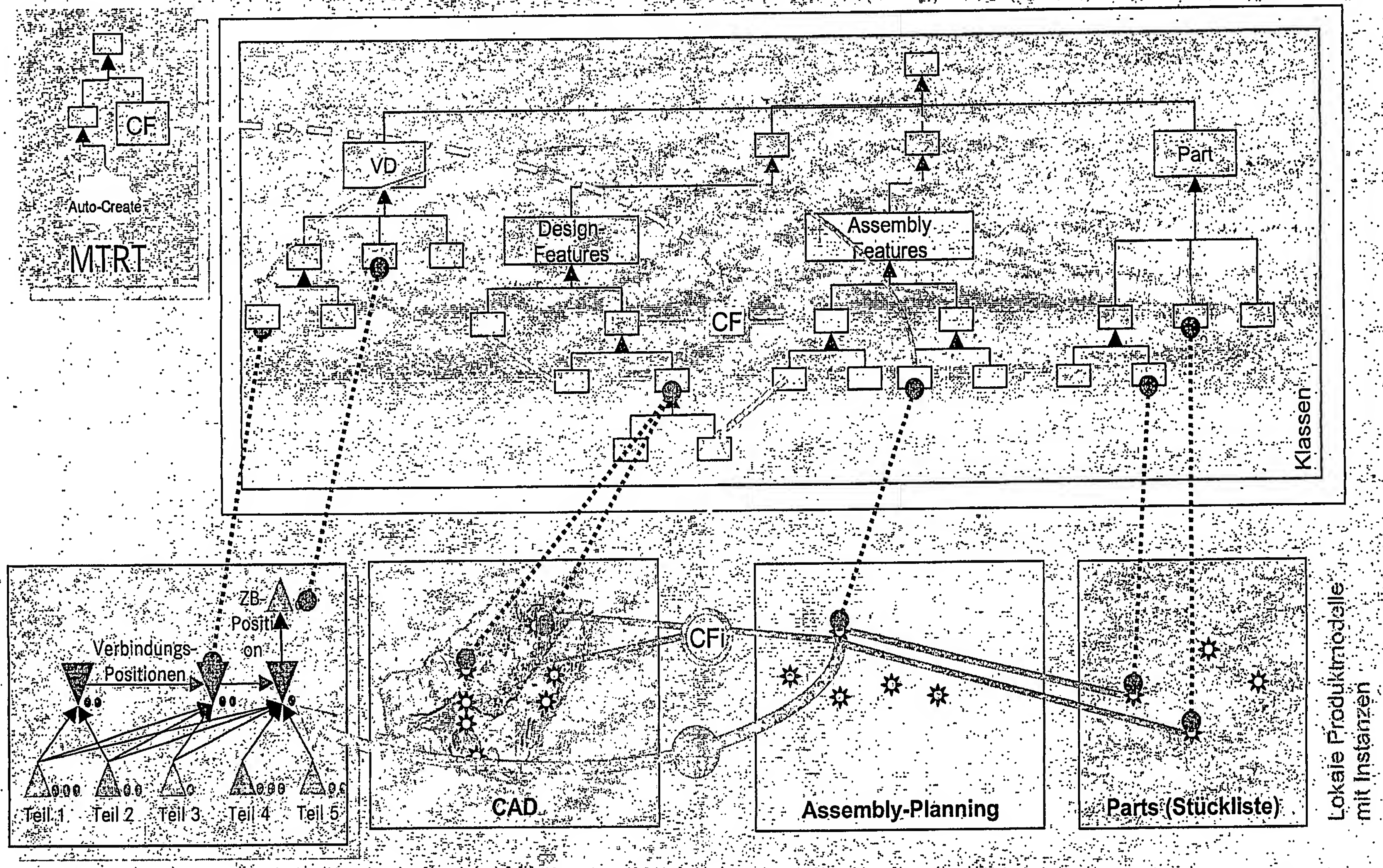


H. Zimmermann, RIC/EP

DAIMLERCHRYSLER

Forschung & Technologie

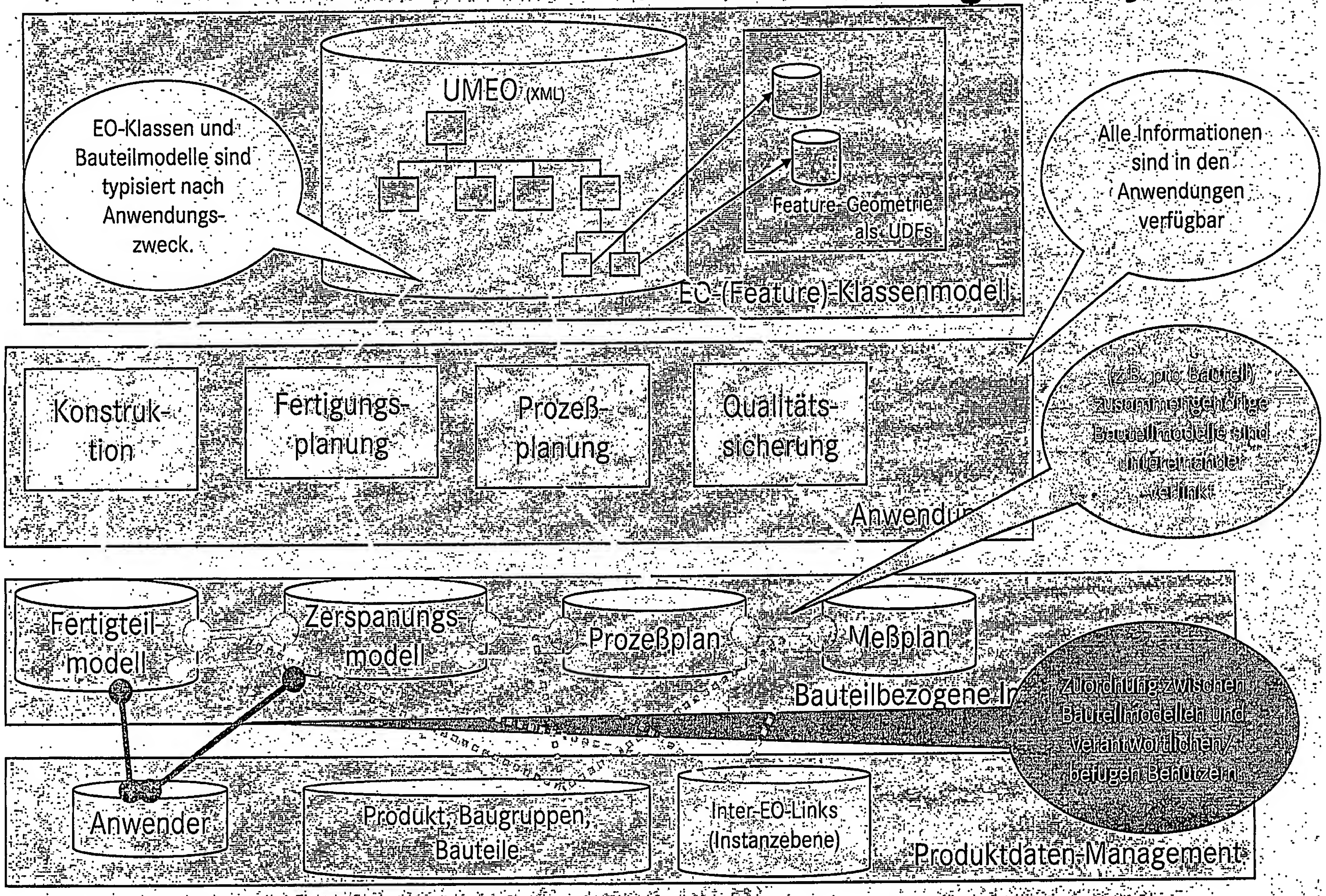
Prinzipschaubild: Verbindungen in ULEO



H. Zimmermann, RIC/EP

DAIMLERCHRYSLER Forschung & Technologie

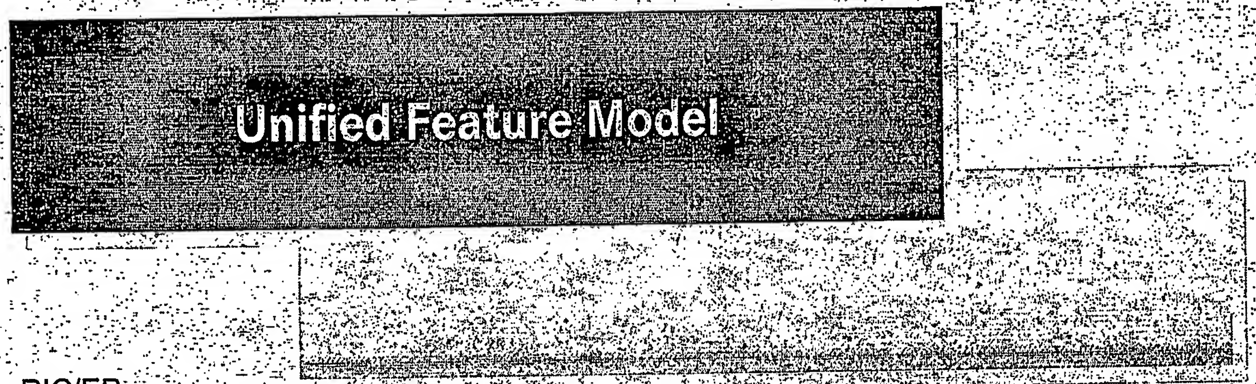
ULEO: zukünftige Zielsysteme


EO-Klassen und Bauteilmodelle sind typisiert nach Anwendungs-zweck.
UMEO (XML)
Feature-Geometrie als UDFs
EO-(Feature)-Klassenmodell
Alle Informationen sind in den Anwendungen verfügbar
Konstruk-tion
Fertigungs-planung
Prozeß-planung
Qualitäts-sicherung
Anwendun
(z.B. pro Bauteil) zusammengehörige Bauteilmodelle sind untereinander verlinkt
Fertigteil-modell
Zerspanungs-modell
Prozeßplan
Meßplan
Bauteilbezogene In
Zuordnung zwischen Bauteilmodellen und verantwortlichen/ befugten Benutzern
Anwender
Produkt, Baugruppen, Bauteile
Inter-EO-Links (Instanzebene)
Produktdaten-Management

H. Zimmermann, RIC/EP

From a Unified Concept Model to a Unified Feature Model

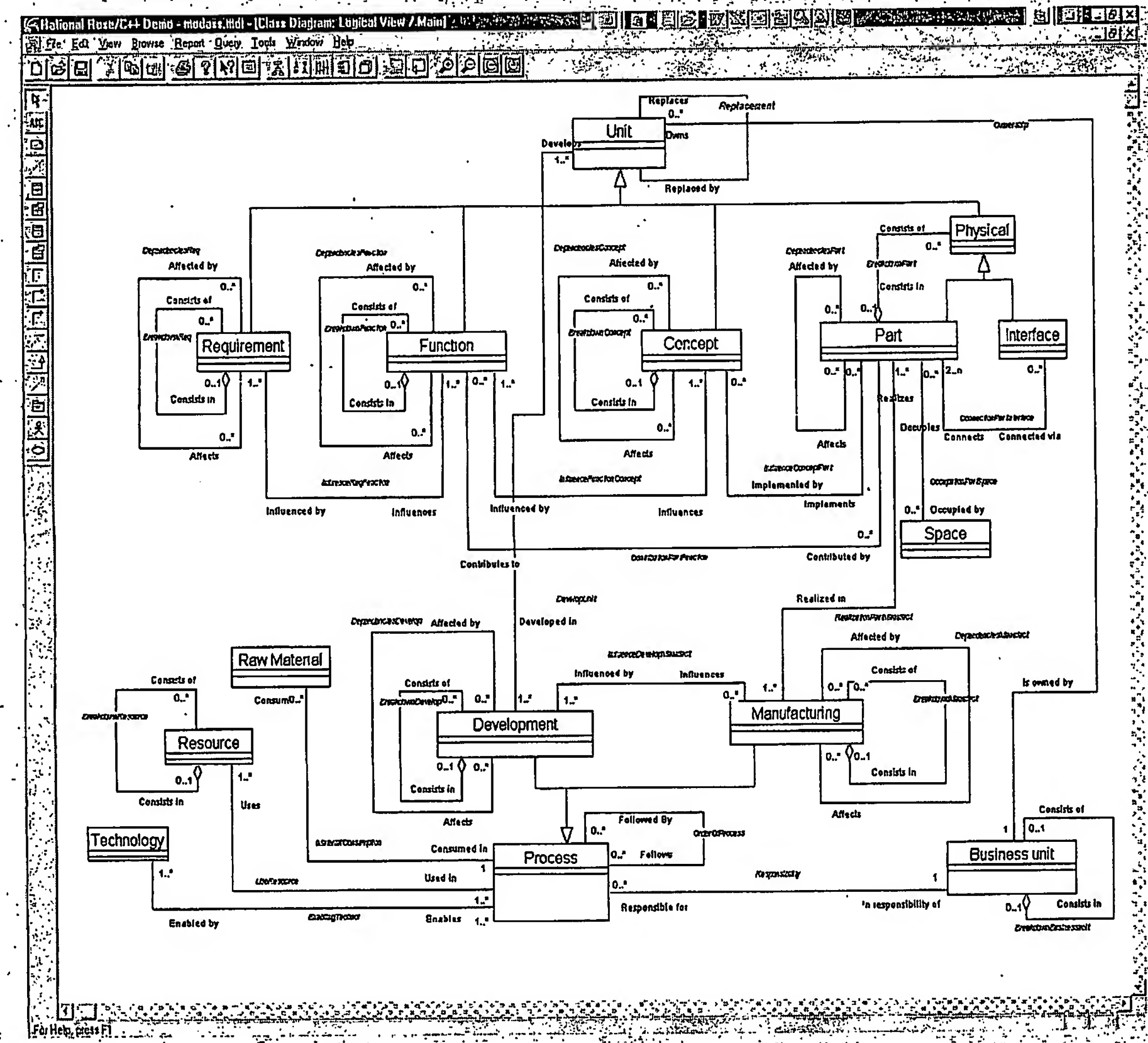
- Object Models need a common understanding of the basic concepts and basic relations between them (also non-hierarchical).
- A Unified Feature Model needs a common understanding of the concepts and relations in product development, manufacturing and logistics.
- Inference tools can be designed using the UCM.
- ⇒ Feature Classes should refer to concepts in a Unified Concept Model to clarify their semantics and behavior.



DAIMLERCHRYSLER

Forschung & Technologie

RIC/EK-Upper Structure

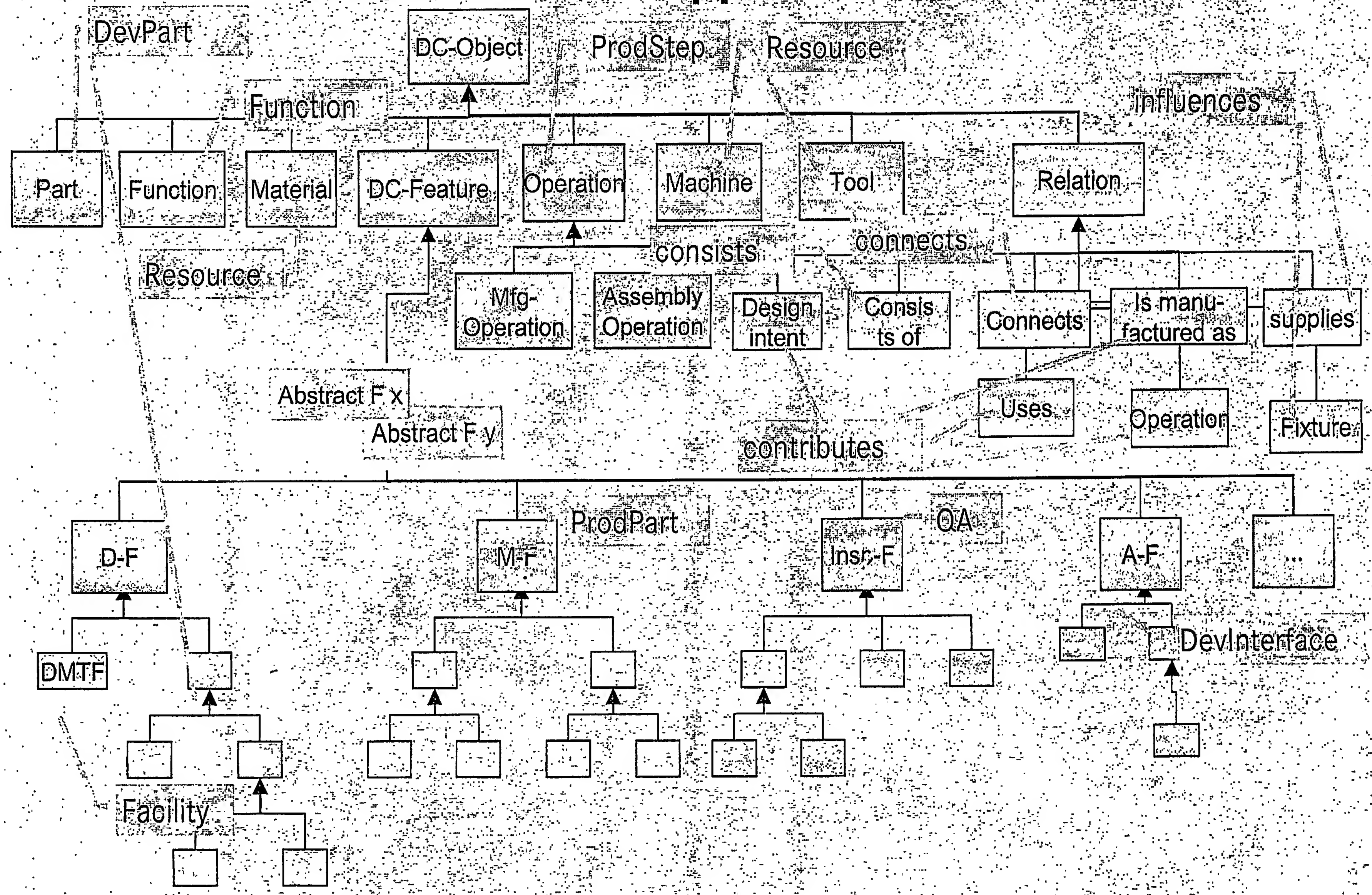


- Basiskonzepte und mögliche Relationen zwischen ihnen zur Modellierung von Produktentwicklung, -herstellung und Logistik.
- UMEO-Klassen könnten auf diese Konzepte referieren.

H. Zimmermann, RIC/EP

DAIMLERCHRYSLER Forschung & Technologie

UMEO und Referenzen auf Upper Structure



H. Zimmermann, RIC/EP